计算机系统能力培养实践课程系列

《微型机系统与接口技术》
实验讲义

——基于 Minisys 平台

Vivado 2017.4

S86 版本 1.4

东南大学计算机系统能力培养
课题组编著

东南大学计算机科学与工程学院、软件学院

2018 年

本版本手册编著人员：

杨全胜、徐言、江仲鸣、蔡玉彤、许恒煜、杨英豪

目 录

第 1 章　Minisys 实验板介绍 1
1.1　Minisys 实验板概述 1
1.1.1　主芯片 XC7A100T 关键资源 1
1.1.2　Minisys 实验板资源 1
1.2　Minisys 板上存储器 3
1.2.1　DDR3 SDRAM 3
1.2.2　SRAM 3
1.2.3　非易失串行 Flash 4
1.3　时钟 4
1.4　基本 I/O 设备 4
1.4.1　拨码开关与 LED 灯 4
1.4.2　按键开关 4
1.4.3　4×4 矩阵键盘 5
1.4.4　七段数码管 6
1.4.5　VGA 模块 8
1.4.6　蜂鸣器 8
1.4.7　麦克风 9
第 2 章　Minisys-S86 系统介绍 10
2.1　Minisys-S86 系统总体结构 10
2.2　S86 处理器 11
2.2.1　S86 处理器内部结构 11
2.2.2　S86 处理器接口信号 12
2.3　内存 13
2.3.1　内存地址空间 13
2.3.2　内存连接 14
2.3.3　内存控制器接口信号 14
2.4　地址译码器 15
2.4.1　I/O 地址空间 15
2.4.2　地址译码器连接示意图 16
2.5　外部控制逻辑 16
2.5.1　读写信号发生器 16
2.5.2　输入数据多路选择器 17
2.5.3　应答信号发生器 18
2.6　Minisys-S86 工具软件的安装和使用 19
2.6.1　i8086 集成开发环境的使用 19
2.6.2　DEBUG 主要命令 22
第 3 章　8086 汇编实验（For PC） 26
3.1　汇编语言程序上机过程 26
3.1.1　Sample 程序的汇编、链接与调试 26
3.1.2　字符串的输入与输出 29
3.2　顺序程序设计实验 29

3.2.1 四则运算.....29
3.2.2 查表的平方值.....30
3.2.3 非压缩 BCD 码转二进制.....30
3.3 分支程序设计实验.....31
3.3.1 简单分支程序设计.....31
3.3.2 数的十进制输出.....32
3.4 循环程序设计.....33
3.4.1 在字符串中替换字符.....33
3.4.2 统计字符串中字符个数.....34
3.5 子程序设计.....34
3.5.1 十进制到十六进制转换.....34
3.5.2 计算 N!.....35
3.6 综合实验.....35
3.6.1 判断回文.....35
3.6.2 有序数组中找数.....35
3.6.3 带反馈输入十进制数字的二进制数出.....36
第 4 章 S86 汇编实验（For Minisys）.....37
4.1 预备知识.....37
4.1.1 使用 S86_SimpleSys.....37
4.1.2 S86_SimpleSys 系统功能调用.....40
4.1.3 汇编编程模板.....41
4.2 顺序程序设计.....42
4.2.1 读取拨码开关的数据输出到 LED.....42
4.2.2 两数相加.....42
4.2.3 查表求平方并以十六进制输出.....43
4.3 分支与循环程序设计.....44
4.3.1 二进制输入，十进制输出.....44
4.3.2 十进制数的输入与输出.....45
4.3.3 两个数的加减乘.....45
4.4 子程序设计.....45
4.4.1 判断回文.....46
4.4.2 利用递归程序，计算 N!。.....46
4.5 综合实验.....46
4.5.1 四则运算计算器.....46
4.5.2 猜数游戏.....47
第 5 章 S86 接口实验.....49
5.1 S86 系统构建及地址译码器设计.....49
5.2 数码管实验.....81
5.2.1 1 位 7 段数码管的设计.....81
5.2.2 4 位 7 段数码管的设计.....82
5.3 4×4 键盘的实验.....86
5.4 计算器实验.....90

5.5　8 位类 8254 定时/计数器设计 92
5.6　可编程中断控制器（PIC）设计 104
5.7　类 8255 可编程并行接口设计 111
5.8　交通灯控制实验 121
5.8.1　红黄绿三色交通灯控制 121
5.8.2　带倒计时的交通灯控制。 123

第 1 章 Minisys实验板介绍

本实验统一采用 Minisys 实验板作为实验平台，所以本章着重介绍 Minisys 的相关功能及参数。

1.1　Minisys实验板概述

Minisys 实验板是一个以 Xilinx Artix-7™ 系列 FPGA（XC7A100T FGG484C-1）为主芯片的可用于“数字电路”、“组成原理”、“微机接口”、“SoC 设计”等多门课程的实验平台。

1.1.1　主芯片XC7A100T关键资源

XC7A100T 上有 101440 个逻辑单元，15850 个 Slice，每一个 Slice 中带有 4 个 6 输入的查找表和 8 个触发器，片内近 12.5%的查找表可以配置为 64-bit 分布式 RAM 或者 32 位的 SRL（或两个 16 位 SRL16），使得综合工具能够充分利用这些逻辑和存储资源。

片内集成 135 个 36Kbit 的 Block RAM，并且每一个可以当作两个独立的 18Kbit 的 Block RAM 使用。这些 Block RAM 资源可以利用 Vivado 的 IP 集成器很方便地配置成单端口、双端口等多种类型 RAM。

240 个 DSP48E1 数字信号处理单元，每个 DSP48E1 中包含一个预加器，一个 25×18 乘法器，一个加法器以及一个累加器。

6 个时钟管理模块（CMT），每个包含 1 个混合模式时钟管理器（MMCM）及一个锁相环（PLL）。MMCM 和 PLL 的中心都有一个可以根据输入电压而调速的晶振，由此能够生成很宽频率范围的时钟信号。同时，这两个部件又都能作为输入时钟信号的抖动滤波器。

内部时钟最高可达 450MHz，Minisys 实验板采用 100MHz 主频。

1.1.2　Minisys实验板资源

Minisys 板卡提供了丰富的端口和外设资源，包括：

- 24 个用户可用的拨码开关
- 5 个按键开关
- 8 个用户可用的红色 LED（RLDs）
- 8 个用户可用的黄色 LED（YLDs）
- 8 个用户可用的绿色 LED（GLDs）
- 2 个 4 位 7 段数码管
- 1 个 4x4 键盘
- 1 个 USB_JTAG 串口通信接口
- 1 个 Micro SD 卡槽
- 1 个 12 位 VGA 输出
- 1 个蜂鸣器
- 1 个 10/100/1000Mbps 以太网接口
- 1 个 512MB 的 DDR3 SDRAM
- 1 组 48bit 位宽，24Mbit 容量的 SRAM

- 1 个 16MB 的非易失性 SPI Flash
- 1 个接口板连接器
- 1 个 6-Pin JTAG 插口，用于 FPGA 的配置及在线调试

大量的 FPGA 资源以及板上的外设让 Minisys 能够胜任各种数字系统设计、计算机组成课程设计等。

Minisys 实验板与 Xilinx 新推出的高性能 Vivado 开发套件（包含 Simulator、IP Integrator、Block Design、SDK）完全兼容。 Xilinx 为这些工具提供了免费的“Webpack”版本，这就意味着可以在不增添费用的情况下完成设计。

图 1-1 是 Minisys（2016 版）实验板实物图。

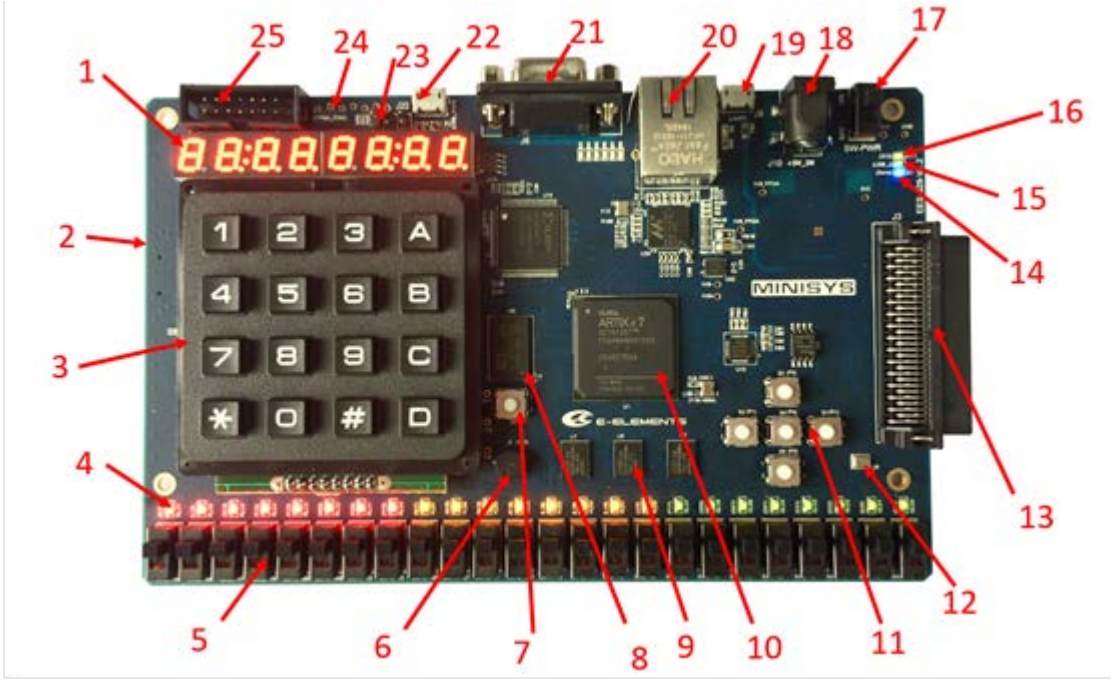


图 1-1 Minisys（2016 版）实验板图

表 1-1 是 Minisys（2016 版）实验板资源一览表。

表 1-1 Minisys（2016 版）实验板资源一览表

标注	描述	标注	描述
1	8 个 7 段数码管	14	FPGA 烧写完成指示灯
2	Micro SD 卡槽(板卡背面)	15	USB_JTAG 指示灯
3	4*4 小键盘	16	电源指示灯
4	LEDs（红、黄、绿各 8 个）	17	电源开关
5	拨码开关	18	电源连接口
6	蜂鸣器	19	USB 转 UART 接口（供电用）
7	FPGA 复位按键	20	以太网接口
8	DDR3 SDRAM	21	VGA 接口
9	SRAM	22	USB_JTAG 接口（编程用）
10	XC7A100T 主芯片	23	编程跳线
11	5 个按键开关	24	用户扩展 I/O
12	麦克风	25	JTAG 接口
13	接口板连接器		

图 1-2 是 Minisys（2017 版）实验板实物图。

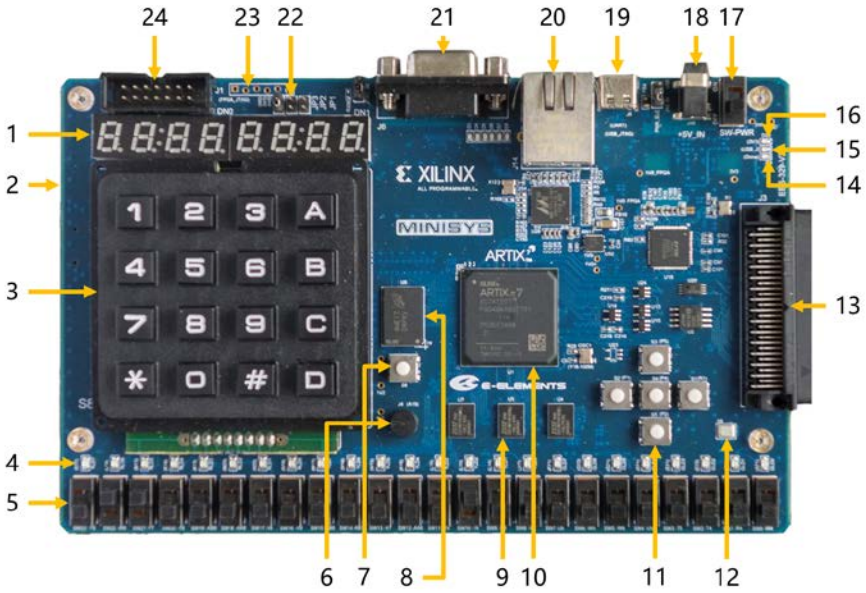


图 1-2 Minisys（2017 版）实验板图

表 1-2 是 Minisys（2017 版）实验板资源一览表。

表 1-2 Minisys（2017 版）实验板资源一览表

标注	描述	标注	描述
1	8 个 7 段数码管	13	接口板连接器
2	Micro SD 卡槽(实验板背面)	14	FPGA 烧写完成指示灯
3	4×4 小键盘	15	USB_JTAG 指示灯
4	LEDs（红、黄、绿各 8 个）	16	电源指示灯
5	拨码开关	17	电源开关
6	蜂鸣器	18	电源连接口
7	FPGA 复位按键	19	Type C 接口（编程，串口）
8	DDR3 SDRAM	20	以太网接口
9	SRAM	21	VGA 接口
10	XC7A100T 主芯片	22	编程跳线
11	5 个按键开关	23	用户扩展 IO
12	麦克风	24	JTAG 接口

1.2　Minisys板上存储器

除了 XC7A100T-1FGG484C 内部有 Block RAM 存储器外，Minisys 板卡上还包括三种扩展存储设备。

1.2.1　DDR3 SDRAM

Minisys 板卡上，将一个容量为 256M×16bit 的 DDR3 SRAM(芯片型号为 MT41J256M16-FBGA96)连接到主芯片上。当主芯片访问 SDRAM 时，需要传送 15 位的行地址、3 位块地址和 10 位列地址，其中行地址和列地址分时共用一组地址线。

1.2.2　SRAM

SRAM 模块由三块 IS61WV51216BLL-10MI 芯片并联组成，每块芯片的容量为 512K×16bit，并联后的 SRAM 模块的容量为 512K×48bit，通过 19 根地址线和 48 根数据线

与主芯片连接。SRAM 的片选信号 $\overline{CE}$、输出使能信号 $\overline{OE}$、写使能信号 $\overline{WE}$、低字节控制 $\overline{LB}$和高字节控制 $\overline{HB}$均为低电平有效。

1.2.3　非易失串行Flash

非易失串行 Flash 的容量是 128Mbits，使用的是专用的 Quard SPI 总线。FPGA 的配置文件可以写入 Quad SPI Flash（型号 N25Q032A13ESE40F），表 1-1 标注 17 的表现最左端两个接上跳帽后选择板子在上电时，FPGA 自动从 SPI Flash 中读取配置文件。当编程跳线连接 JP3 的位置时，可以将编程文件下载到 Flash 中。

1.3　时钟

Minisys 板卡包括了一个连接在主芯片 Y18 管脚的 100MHz 的晶振。根据需求设计，输入时钟可以驱动 MMCMs 或 PLLs 产生多种频率的时钟以及相位的变化。Xilinx 提供了时钟向导 IP 核可以帮助用户设计产生不同需求的时钟。

1.4　基本I/O设备

1.4.1　拨码开关与LED灯

Minisys 实验板上有 24 个拨码开关，其板上标注为 SW23~SW0。在实验中，常将拨码开关作为数据输入，当开关拨到下档时，表示输入为 0，否则为 1。

另外，实验板上还有 24 个 LED 灯（红、黄、绿分别 8 个），板上标号为 RLD7~0、YLD7~0 和 GLD7~0。当 FPGA 相应管脚的输出为高电平时，所连接的 LED 灯被点亮，否则灯熄灭。

拨码开关和 LED 灯与主芯片的连接如图 1-3 图 1-3 拨码开关、LED 灯电路图所示。

1.4.2　按键开关

按键开关与主芯片的连接方式如图 1-4 所示。当某一按键按下时，其对应的 FPGA 输入为 1，否则为 0。

板上共有 6 个按键开关（S1~S6），其中的 S6 按键被选作 FPGA 的复位按键。在开发学习过程中，建议在需要时设置一个复位输入，这不仅是所开发系统的功能需求，还有利于代码调试。

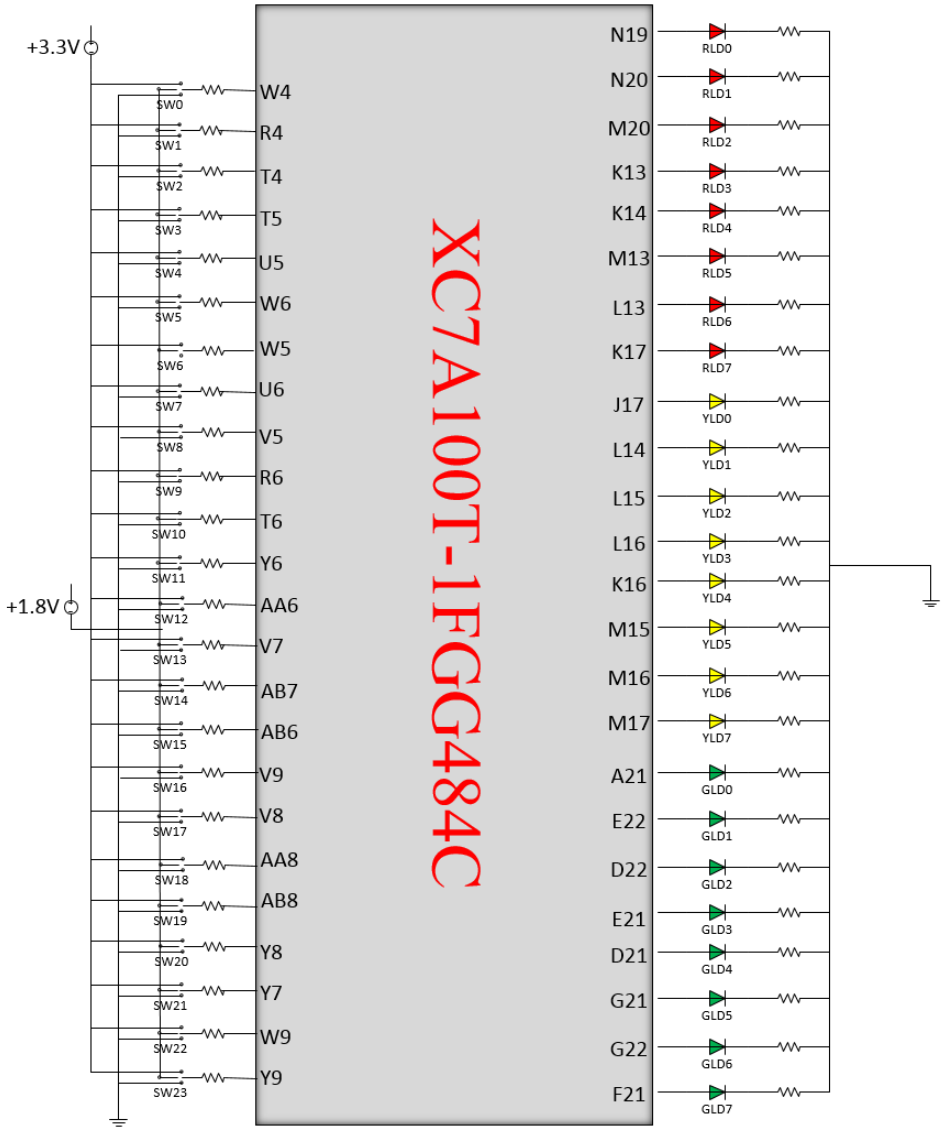


图 1-3 拨码开关、LED 灯电路图

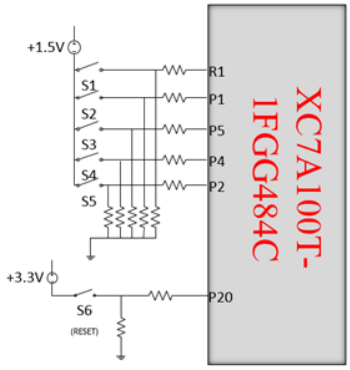


图 1-4 按键开关电路图

1.4.3　4×4 矩阵键盘

Minisys 实验板上还连接了一个 4×4 的矩阵键盘，用以方便快速的进行数值输入。

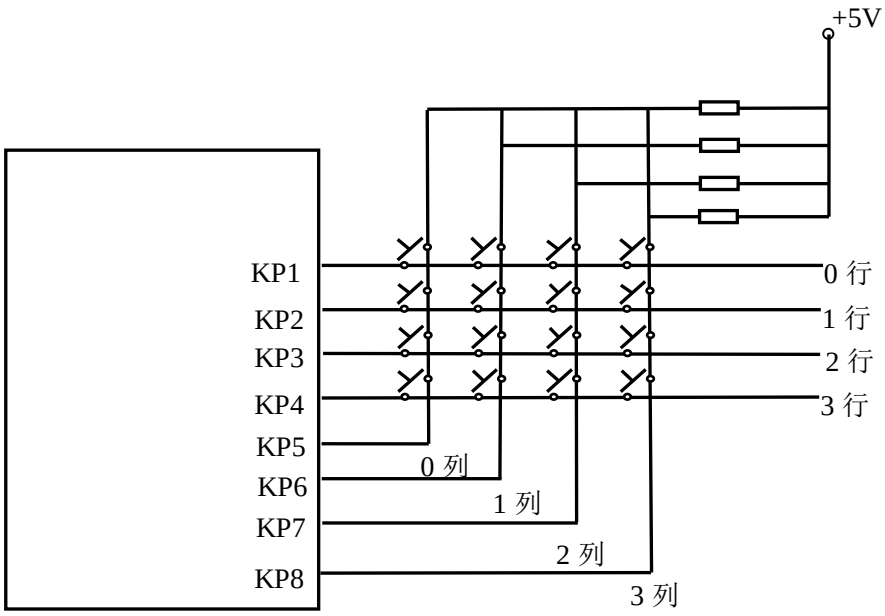


图 1-5 4×4 矩阵键盘原理图

4×4 键盘通过 4 根行选线和 4 根列选线连接到主芯片。其采用行列扫描的原理与主芯片交换数据，图 1-5 显示了 4×4 键盘的原理图。值得注意的是，主芯片接收的是按键的“坐标”，而不是其所对应的键值。要想获得需要的键值，需要在程序中对行、列信号的每一种组合方式进行翻译。

图 1-6 显示了 Minisys 实验板上的 4×4 键盘与主芯片的连接方式。

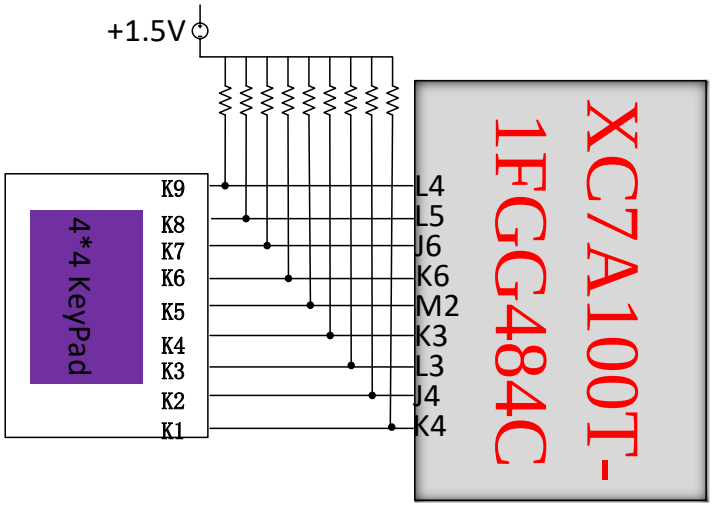


图 1-6 4×4 键盘连接电路图

1.4.4　七段数码管

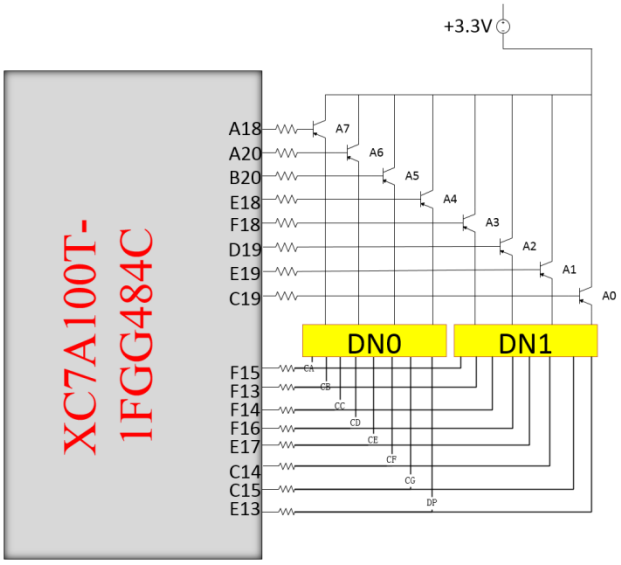


图 1-7 七段数码管电路图

Minisys 实验板上有两个 4 位带小数点的七段数码管，图 1-7 显示了它们与主芯片的连接方式。其中 A7~A0 是数码管 8 个位的使能信号，而 CA~CG/DP 则对应各个位上七个段以及小数点的触发信号。需要注意的是，使能信号和触发信号都是低电平有效。

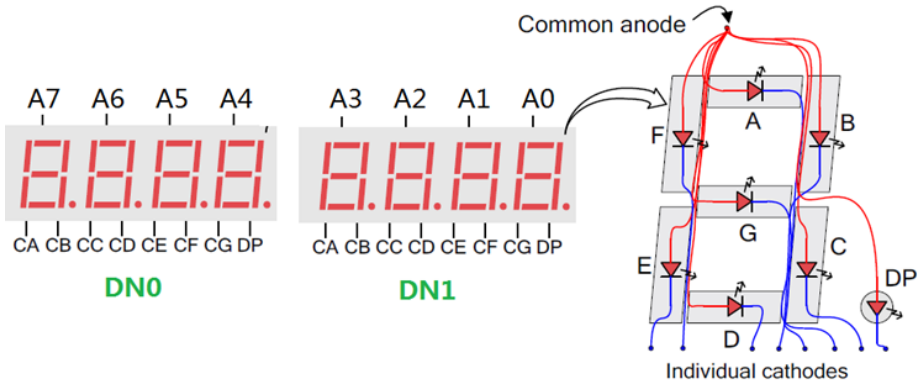


图 1-8 共阳极数码管电路结构

图 1-8 以数码管中最右侧的 A0 数码管为例说明了 Minisys 板卡上的 7-段数码管的连接方式。8 个位中的各个相应的段及小数点分别连接到一组低电平触发的引脚上，他们被称为 CA、CB、CC、…、CG、DP，其中，CA 接到这 8 个数码管中每一个数码管 A 段的负极,CB 接到这 8 个数码管中每一个数码管 B 段的负极，以此类推。

此外，每一个数码管都有一个使能信号 A[7:0]。A[7:0]通过一个反相器接到对应数码管的每一个段的正极上。比如说，只有当 A[0]为 0 的时候，最右侧数码管的显示才会受到 CA…CG 这几个信号的驱动。

图 1-9 中列出了数码管显示 0 到 F 时点亮的段。比如说在显示数字 0 的时候，除了中间的 G 段外其他的段都被点亮了，而数字 1 只点亮了 B 段和 C 段。

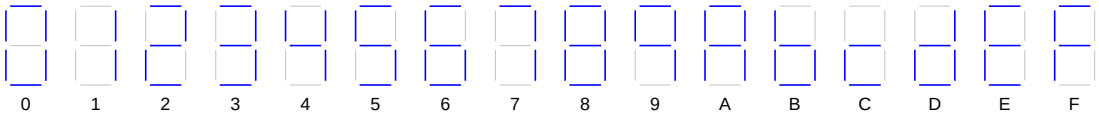


图 1-9 7-段数码管显示功能

要想让每个数码管显示不同的数字，使能信号（A[7:0]）和段信号（CA…CG）必须依次地被持续驱动，数码管之间的刷新速度应该足够快这样就看不出来数码管之间在闪烁，通常刷新频率可以设置为 500Hz，也就是 2ms 刷新一次。举个例子，如果想在数码管 0 上显示数字 3 而数码管 1 上显示数字 9，可以先把 CA…CG 设置为显示数字 3，并拉低 A[0]信号，2ms 后再把 CA…CG 设置为显示数字 9 并拉高 A[0]拉低 A[1]。

1.4.5　VGA模块

VGA 模块与主芯片的连接电路图如图 1-10 所示。Minisys 使用 14 路 FPGA 信号生成一个 VGA 端口，该端口包括 4 位的红、绿、蓝三基色信号和标准行、列同步信号。色彩信号由电阻分压电路产生，支持 12 位的 VGA 彩色显示，共有 4096 种不同的颜色配置。

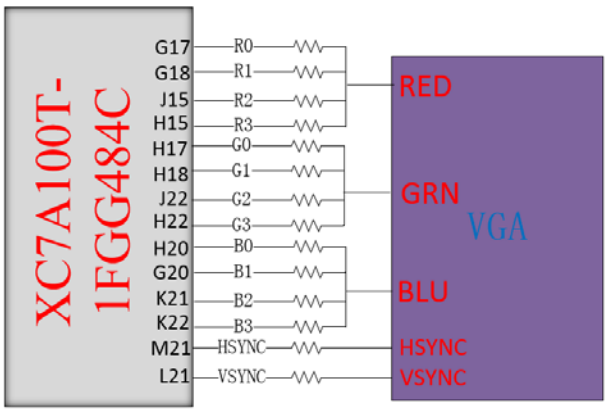


图 1-10 VGA 模块电路图

在实际应用中，若使用 Minisys 实验板进行 VGA 显示输出，一定要确保提供正确的时序，否则 VGA 显示电路不能正常工作。

1.4.6　蜂鸣器

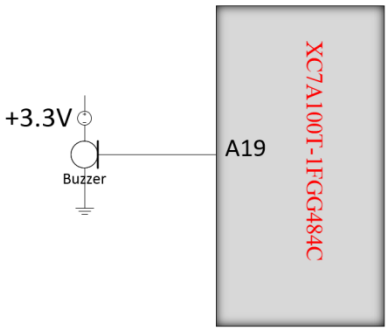


图 1-11 蜂鸣器连接电路图

除了上述的各种视觉输出部件，Minisys 实验板上还配置了一个蜂鸣器用作声音输出部件。与主芯片的连接方式如图 1-11 所示。主芯片通过 A19 管脚向蜂鸣器输出一个电信号，该信号的频率由用户决定。在该信号驱动下，蜂鸣器内部发生机械振动，发出相应频率的声音。

1.4.7　麦克风

Minisys 实验板上还包含一个全向的 MEMS 麦克风。麦克风使用 ADMP421 芯片，其信噪比高达 61dBA，敏感度达-26dBFS，能够对 100Hz~15kHz 的信号产生平稳的响应，经其数字化后的音频以 PDM 格式输出。图 1-12 显示了麦克风模块与主芯片的连接方式。

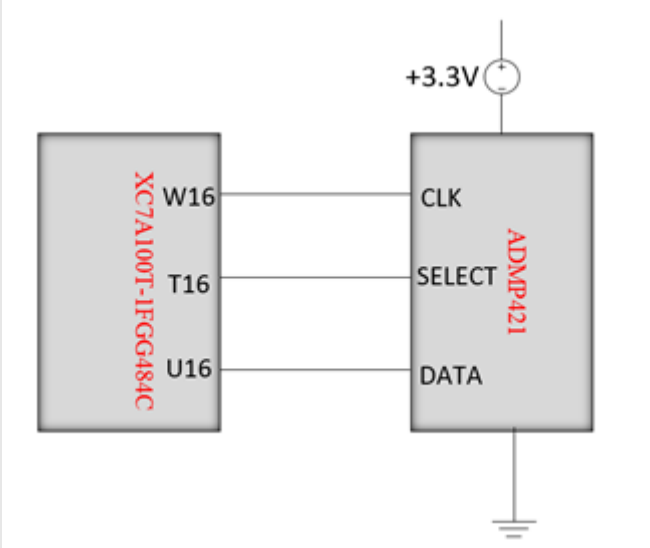


图 1-12 麦克风连接电路图

第 2 章 Minisys-S86 系统介绍

本章介绍 Miisys-S86 系统的硬件架构和集成开发环境 i8086 IDE 的安装和使用，对于 Xilinx 的开发工具 Vivado 的使用请读者参考 Xilinx 的有关文档。

2.1　Minisys-S86 系统总体结构

Minisys-S86 系统主要由时钟、复位、S86 处理器、内存、地址译码器、读写信号发生器、输入数据多路选择器、可编程中断控制器 PIC 和各个外设构成，并通过 wishbone 总线连接。系统总体结构如

图 2-1 所示。

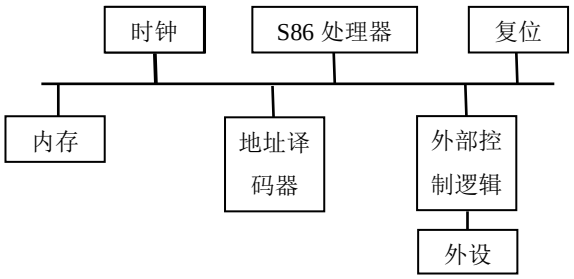


图 2-1 Minisys-S86 系统总体结构图

时钟：系统的时钟，由板上 100MHZ 时钟经过 PLL 分频得到，其中 CPU 时钟为 10MHZ，内存时钟为 30MHZ。由于 Minisys-S86 系统使用的内存为 FPGA 自带的 Blockram，而 Blockram 实现上依靠时钟读写，当采用 CPU 相同频率的时钟时会产生读写上的滞后，所以需要额外使用一个高频率时钟。

复位：系统的复位，当板上的复位按键按下或者 PLL 的 lock 信号为低时复位。

S86 处理器：系统的核心，主要负责数据处理、控制与传送；

内存：存放指令和数据，根据内存读写信号和地址信号读出数据至处理器或者写入数据；

地址译码器：通过判断 I/O 读写信号和译码高位地址来产生片选信号；

外部控制逻辑：主要包括读写信号发生器、输入数据多路选择器、应答信号发生器和 PIC。读写信号发生器根据从 S86 处理器输出的控制信号来产生 I/O 读写信号和内存的读写信号；输入数据多路选择器根据控制信号和片选信号从内存、中断向量号和多个外设的输入数据中选择相应的输入数据输入至处理器；应答信号发生器根据内存的应答信号和各个外设的片选信号产生应答信号发送给 S86 处理器；PIC 可以通过编程进行中断配置，处理中断请求产生 INTR 信号和中断向量号，并根据 INTA#信号撤销 INTR 信号；

外设：各个外设根据片选信号、读写信号和低位地址信号将数据通过输入数据多路选择器输入至处理器或者从处理器获取数据。

除去时钟和复位，主要组成单元的具体细节如图 2-2 所示。

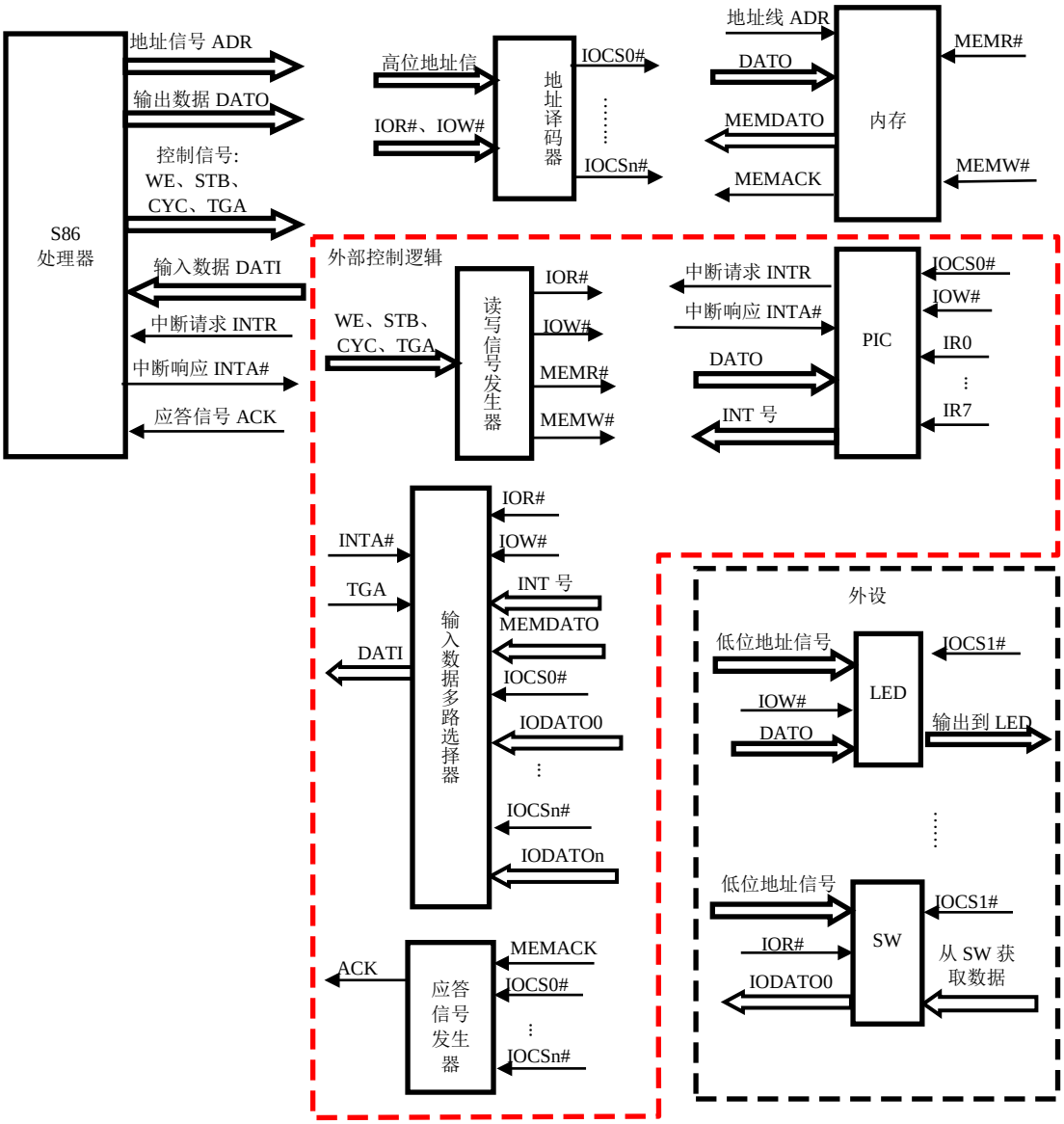


图 2-2　Minisys-S86 系统总体结构细节图

2.2　S86 处理器

本讲义所使用的 S86 处理器是在开放源码的 Zet Processor 基础上，根据接口实验的需要进行了一些调整和扩充而来。Zet Processor（http://zet.aluzina.org/index.php/Zet_processor）是一个基于 IA-32 架构(x86）的开源应用，它可以订制为 ASIC，也可以在可编程器件如 CPLD 或 FPGA 上实现。

2.2.1　S86 处理器内部结构

S86 处理器内部结构如图 2-3 所示，主要由指令译码器、微指令控制器、执行单元、有限状态机、总线控制器组成，数据总线宽度为 16 位，地址总线宽度为 20 位。

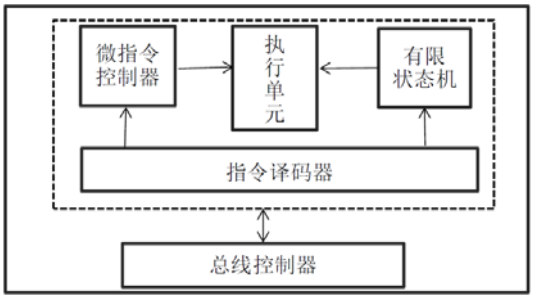


图 2-3 S86 处理器内部结构

S86 处理器主要在三个方面区别于 X86 处理器，如表 2-1 所示。

表 2-1 S86 处理器与 X86 处理器的区别

区别	S86	X86
1	数据线、地址线不复用，没有地址锁存信号	数据线和地址线复用，有地址锁存信号
2	数据线是单向的，分为输入数据线和输出数据线	数据线是双向的
3	一次有效的 INTR 会产生一次 INTA#	一次有效的 INTR 会产生两次 INTA#

2.2.2　S86 处理器接口信号

Minisys-S86 系统所使用的为 wishbone 总线，该总线具有如下特点：简单、紧凑、需要很少的逻辑门。 S86 处理器的相应接口信号定义如表 2-2 所示。

表 2-2 S86 处理器接口信号

信号名称	位宽	类型	意义
wb_clk_i	1	输入	时钟信号
wb_rst_i	1	输入	同步复位信号
wb_dat_i	16	输入	数据信号（输入）
wb_dat_o	16	输出	数据信号（输出）
wb_adr_o	19	输出	地址信号
wb_we_o	1	输出	读/写信号
wb_tga_o	1	输出	地址标签信号
wb_sel_o	2	输出	有效数据总线选择信号
wb_stb_o	1	输出	选通信号
wb_cyc_o	1	输出	总线周期信号
wb_ack_i	1	输入	操作结束方式信号
wb_tgc_i	1	输入	总线周期标签信号（输入）
wb_tgc_o	1	输出	总线周期标签信号（输出）

各个信号具体功能描述如下：

1、wb_clk_i：时钟信号。S86 处理器的基本时钟，由板子上 100MHZ 的时钟引脚经过 PLL 锁相环分频到 10MHZ 得到。

2、wb_rst_i：同步复位信号。高电平有效，当板上复位按键按下或者 PLL 的 lock 信号为低电平时有效。

3、wb_dat_i：数据信号（输入）。输入 S86 处理器的数据，从输入数据多路选择器中获取，包括读取的内存数据和指令，中断向量号和读取的外设数据。

4、wb_dat_o：数据信号（输出）。从 S86 处理器输出的数据，包括写入内存的数据和写给外设的数据。

5、wb_adr_o：地址信号。从 S86 处理器中输出的地址，一般为访问内存的地址或者外设的地址。此处地址信号只有 19 位，因为只提供偶数地址访问，所以第 0 位总是 0，为了防止非法访问，故第 0 位不提供外部连接。

6、wb_we_o：读/写信号。用以表示读或者写操作，其中高电平表示写，低电平表示读。送至读写信号发生器产生 I/O 读写信号和内存读写信号。

7、wb_tga_o：地址标签信号。附加于地址总线上的标签，用以区分访问外设还是内存，其中高电平表示访问外设，低电平表示访问内存。

8、wb_sel_o：有效数据总线选择信号。标识当前操作中数据总线上哪些比特是有效的，共两位，高电平表示数据有效。

9、wb_stb_o：选通信号。高电平有效，有效表示 S86 处理器发起一次总线操作。只有选通信号有效（此时 wb_cyc_o 也必须为高）时，wb_adr_o、wb_dat_i、wb_dat_o 才有意义。

10、wb_cyc_o：总线周期信号。高电平有效，有效表示 S86 处理器请求总线的使用权或者正在占有总线，但是不一定正在进行总线操作（是否正在进行总线操作取决于选通信号 wb_stb_o），是最高层的控制信号，只有该信号有效其他信号才有意义。

11、wb_ack_i：操作结束方式信号（应答信号）。高电平表示总线操作成功，即收到操作成功的应答，从应答信号发生器获取。

12、wb_tgc_i：总线周期标签信号（输入）。此处用做 INTR。

13、wb_tgc_o：总线周期标签信号（输出）。此处用做 INTA。

2.3　内存

Minisys-S86 系统使用的内存为 Minisys 平台所使用的 FPGA 中带有的 Block Ram 资源，共 128KB。

2.3.1　内存地址空间

Minisys-S86 系统的内存地址空间如表 2-3 所示，其中 00000H~003FFH 为 1KB 的中断向量表（系统专用），00400H~007FFH 为 1KB 的系统数据区(BIOS 专用），数据段和栈段占用 00800H~0FFFFH 的 62KB 地址空间（由编程人员进行分配），用户代码段为 F0000H~FEFFFH 共 63KB，最后 1KB 用来存放 BIOS 代码，其中在 FFFEEH~FFFF0H 存放中断返回指令，最后 16B 的地址空间存放的为 BOOT 代码。

表 2-3 内存地址空间

地址空间	意义
FFFFFH FFFF0H	BOOT
FFFEFH FFFEEH	IRET
FFFEDH FFC00H	BIOS 代码段
FFBFFH F0000H	用户代码段
.....	保留
0FFFFH 00800H	用户数据段+栈段
007FFH 00400H	用户数据区 （BIOS 专用）
003FFH 00000H	中断向量表 （BIOS 专用）

2.3.2　内存连接

内存与其他部件的连接示意图如图 2-4 所示。

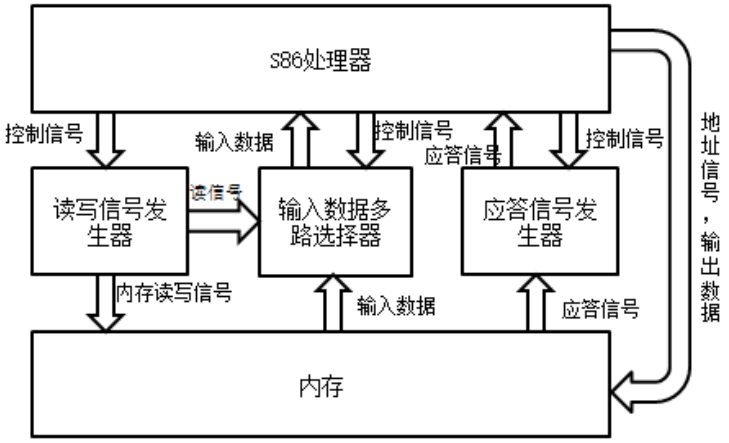


图 2-4 内存与其他部件连接示意图

内存控制器根据从读写信号发生器获取的内存读写信号判断是否进行内存读写。读内存操作，通过从 S86 处理器获取地址信号读取内存数据，并将读取的内存数据通过输入多路数据选择器送至 S86 处理器；写内存操作，通过从 S86 处理器获取地址信号和需写入数据写入数据至内存中。不论读写内存操作，均会产生应答信号给应答信号发生器

2.3.3　内存控制器接口信号

内存通过内存控制器进行控制。内存控制器的相应接口信号定义如表 2-4 所示。

表 2-4 内存接口信号定义

信号名称	位宽	类型	意义
wb_clk_i	1	输入	时钟信号
wb_dat_i	16	输入	数据信号（输入）
wb_dat_o	16	输出	数据信号（输出）
wb_adr_i	19	输入	地址信号
wb_sel_i	2	输入	有效数据总线选择信号
wb_ack_o	1	输出	操作结束方式信号
MEMW_N	1	输入	内存写信号
MEMR_N	1	输入	内存读信号

各个信号具体功能描述如下：

1、wb_clk_i：时钟信号。为内存的基本时钟，由板子上 100MHZ 的时钟引脚经过 PLL 锁相环分频到 30MHZ 得到。

2、wb_dat_i：数据信号（输入）。写内存操作时，写入内存的数据，从 S86 处理器获取。

3、wb_dat_o：数据信号（输出）。读内存操作时，读出内存的数据，通过输入多路选择器输入至 S86 处理器。

4、wb_adr_i：地址信号。从 S86 处理器获取地址。此处地址信号只有 19 位，因为只提供偶数地址访问，所以第 0 位总是 0，为了防止非法访问，故第 0 位不提供外部连接。

5、wb_sel_i：有效数据总线选择信号。由于数据线位数为 16，而 S86 支持 8 位写，写内存操作时，用以区分写哪个字节。

6、wb_ack_o：操作结束方式信号（应答信号）。高电平表示总线操作成功，即发出操作成功的应答，输出至应答信号发生器。

7、MEMW_N：内存写信号。低电平有效，有效表示处于内存写周期，从读写信号发生器获取。

8、MEMR_N：内存读信号。低电平有效，有效表示处于内存读周期，从读写信号发生器获取。

2.4　地址译码器

地址译码器主要对 I/O 地址高位地址进行译码产生 PIC 的片选信号和外设的片选信号，其中 I/O 地址设定为低 10 位有效，高 6 位作为高位地址输入地址译码器。

2.4.1　I/O地址空间

Minisys-S86 系统的 I/O 地址空间如表 2-5 所示，其中 **00000H** 和 **00002H** 分别对应 LED 的低 16 位和高 8 位；**00010H** 和 **00012H** 分别对应 Switch 低 16 位和高 8 位；**00020H** 和 **00022H** 分别对应数码管 0、1，**00024H** 对应数码管控制器；**00030H** 对应 4X4 键盘（低 5 位为有效数据）；**00040H**、**00042H**、**00044H** 分别对应 8254 计数器 0、1 和 2，**00046H** 对应 8254 控制器；**00050H**、**00052H** 对应 PIC；**00060H**、**00062H** 和 **00064H** 分别对应 8255 的 A、B 和 C 口，**00066H** 对应 8255 控制寄存器；**00070H** 和 **00072H** 分别对应交通灯数码管端口和计数值端口。

此处地址安排采用地址线的第 9 位到 4 位作为地址译码的高端地址，第 3 位到第 0 位作为低端地址。

还要注意一点，大家在学 X86 微机原理与接口技术的时候，是以 8088 CPU 作为处理器，由于 8088 对外 I/O 数据线只有 8 位，因此对 I/O 寻址可以是字节寻址，也就是地址可以是奇数也可以是偶数，而 S86 是按照 8086 的设计，I/O 数据线是 16 位，因此其寻址是字边界对齐的，也就是 I/O 地址都是偶数地址。

表 2-5 I/O 地址空间

外设	外设片选地址范围	地址	外设端口地址	类型
LED	00000H~0000FH	00000H	LED 低 16 位	OUT
		00002H	LED 高 8 位	OUT
Switch	00010H~0001FH	00010H	Switch 低 16 位	IN
		00012H	Switch 高 8 位	IN
数码管	00020H~002FH	00020H	数码管 0 组（16 位）	OUT
		00022H	数码管 1 组（16 位）	OUT
		00024H	数码管控制器（16 位）	OUT
4X4 键盘	00030H~003FH	00030H	4X4 键盘（低 5 位有效）	IN
8254	00040H~004FH	00040H	8254 计数器 0	OUT
		00042H	8254 计数器 1	INOUT
		00044H	8254 计数器 2	INOUT
		00046H	8254 控制	INOUT
PIC	00050H~005FH	00050H	PIC 的 ICW1，OCW1	INOUT
		00052H	PIC 的 ICW2，OCW2	INOUT
8255	00060H~006FH	00060H	8255 读写 A 口	INOUT
		00062H	8255 读写 B 口	INOUT
		00064H	8255 读写 C 口	INOUT
		00066H	8255 控制寄存器	IN
交通灯扩展	00070H~007FH	00070H	交通灯数码管端口	IN
		00072H	交通灯计数值端口	IN

2.4.2　地址译码器连接示意图

此处给出地址译码器的连接示意图，如图 2-5 所示，对应的信号定义涉及到实验，请在实验中完成。

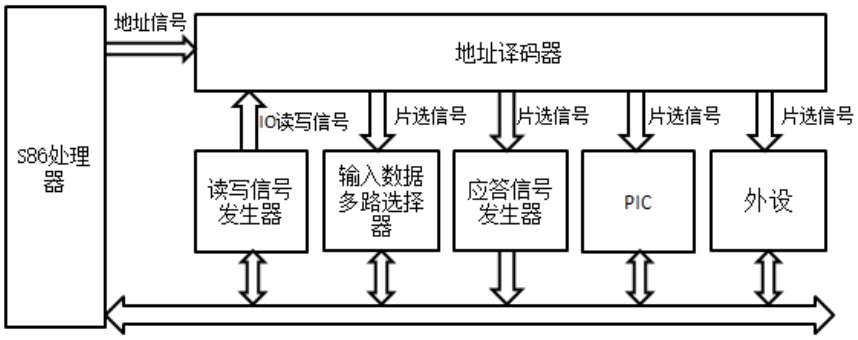


图 2-5 地址译码器连接示意图

地址译码器通过对从读写信号发生器获取的 I/O 读写信号和从 S86 处理器获取的高位地址进行译码产生 PIC 片选信号和给外设的片选信号，而应答信号发生器需要判断片选信号产生应答信号，输入数据多路选择器接受从内存或多个外设中输入的数据，通过对片选信号的判断选择一路数据给 S86 CPU。

2.5　外部控制逻辑

外部控制逻辑主要包括读写信号发生器、输入数据多路选择器、应答信号发生器和 PIC。读写信号发生器产生外设读写信号和内存的读写信号；输入数据多路选择器从多个输入数据中选择一路输入数据给 S86 CPU；应答信号发生器通过片选信号和内存应答信号产生应答信号；PIC 可以设置中断模式，产生中断过程所需时序。

2.5.1　读写信号发生器

读写信号发生器连接示意图，如图 2-6 所示。

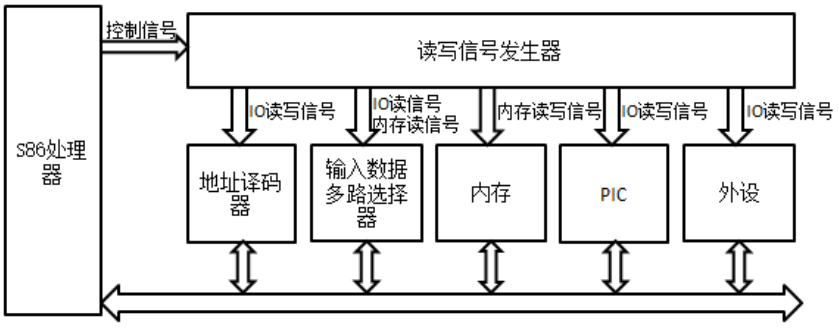


图 2-6 读写信号发生器连接示意图

读写信号发生器通过 S86 处理器获取的控制信号产生 I/O 读写信号和内存读写信号，其中 I/O 读写信号将会连接到地址译码器、PIC、外设和输入数据多路选择器。内存读写信号连接到内存。

读写信号发生器的相应接口信号定义如表 2-6 所示。

表 2-6 读写发生器接口信号定义

信号名称	位宽	类型	意义
wb_tga_i	1	输入	地址标签信号
wb_we_i	1	输入	读/写信号
wb_cyc_i	1	输入	总线周期信号
wb_stb_i	1	输入	选通信号
IOW_N	1	输出	I/O 写信号
IOR_N	1	输出	I/O 读信号
MEMW_N	1	输出	内存写信号
MEMR_N	1	输出	内存读信号

各个信号具体功能描述如下：

1、wb_tga_i：地址标签信号。附加于地址总线上的标签，用以区分访问外设还是内存，其中高电平表示访问外设，低电平表示访问内存。从 S86 处理器获取。

2、wb_we_i：读/写信号。用以表示读或者写操作，其中高电平表示写，低电平表示读。从 S86 处理器获取。

3、wb_cyc_i：总线周期信号。表示 S86 处理器请求总线的使用权或者正在占有总线，只有该信号有效其他信号才有意义。。高电平表示有效，低电平表示无效。

4、wb_stb_i：选通信号。选通有效表示 S86 处理器发起一次总线操作。高电平表示有效，低电平表示无效。

5、IOW_N：I/O 写信号。低电平有效。

6、IOR_N：I/O 读信号。低电平有效。

7、MEMW_N：内存写信号。低电平有效。

8、MEMR_N：内存读信号。低电平有效。

其中 IOW_N、IOR_N、MEMW_N、MEMR_N 有效时，输入信号的值，如表 2-7 所示。

表 2-7 读写信号有效对应输入信号值表

信号名称	有效值	wb_tga_i	wb_we_i	wb_cyc_i	wb_stb_i
IOW_N	0	1	1	1	1
IOR_N	0	1	0	1	1
MEMW_N	0	0	1	1	1
MEMR_N	0	0	0	1	1

2.5.2　输入数据多路选择器

输入数据多路选择器连接示意图，如图 2-7 所示。

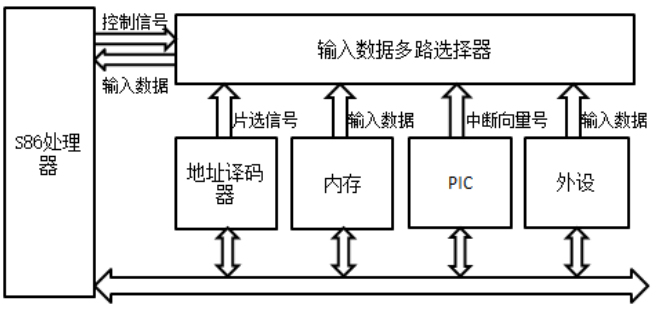


图 2-7 输入数据多路选择器连接示意图

输入数据多路选择器通过从 S86 处理器获取的控制信号和地址译码器产生的片选信号，

从多路的输入数据（内存输入数据、外设输入数据和 PIC 来的中断向量号）中选择一路输入至 S86 处理器。

输入数据多路选择器的相应接口信号定义如表 2-8 所示。

表 2-8 输入数据多路选择器接口信号定义

信号名称	位宽	类型	意义
wb_tga_i	1	输入	地址标签信号
RDN	1	输入	读信号
MEMDATI	16	输入	内存输入数据
IOCS_N	1	输入	I/O 片选信号
IODATI	16	输入	I/O 输入数据
.....			其余 I/O 片选信号和输入数据
DATO	16	输出	输入至 S86 处理器数据

各个信号具体功能描述如下：

1、wb_tga_i：地址标签信号。附加于地址总线上的标签，用以区分访问外设还是内存，其中高电平表示访问外设，低电平表示访问内存。从 S86 处理器获取。

2、RDN：读信号。低电平有效，当发生内存读或者 I/O 读操作时有效，即 IOR_N 和 MEMR_N 中至少有一个为低电平。

3、IOCS_N：I/O 片选信号。低电平有效，由地址译码器产生。

4、IODATI：I/O 输入数据。当发生读外设操作时，由相应的外设产生。

5、DATO：经过多路选择，输入至 S86 处理器的数据。

输入数据多路选择器首先通过 RDN 信号来判断是否有数据要输入至 S86 处理器，然后判断 wb_tga_i 信号，如果为低电平，则选择 MEMDATI 为输入数据，如果 wb_tag_i 为高电平，则依次判断输入的外设的片选信号，有效则选择对应的外设输入数据为输入数据。

2.5.3　应答信号发生器

应答信号发生器连接示意图，如图 2-8 所示。

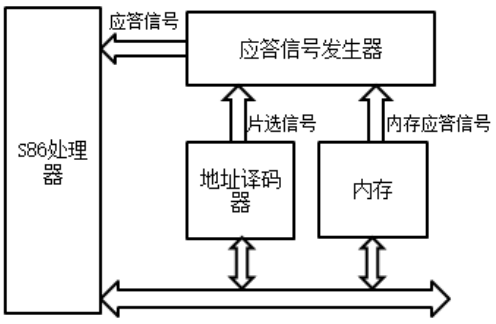


图 2-8 应答信号发生器连接示意图

应答信号发生器首先判断从内存获取的应答信号，如果有效，则产生有效应答，如果无效则依次判断从地址译码器获取的片选信号，如果存在有效片选信号，则发送有效应答，否则产生无效应答。

应答信号发生器的相应接口信号定义如表 2-9 所示。

表 2-9 应答信号发生器接口信号定义

信号名称	位宽	类型	意义
MEMACK	1	输入	内存应答信号
IOCS_N	1	输入	I/O 片选信号
.....			其余的 I/O 片选信号

ACK	1	输出	应答信号

各个信号具体功能描述如下：

1、MEMACK：内存应答信号。高电平有效，有效时应答信号发生器产生有效应答。

2、IOCS_N：I/O 片选信号。低电平有效，有效时应答信号发生器产生有效应答。。

3、ACK：应答信号。当应答信号发生器产生有效应答时变高，输出至 S86 处理器。

2.6　Minisys-S86 工具软件的安装和使用

打开资源包的 C:\sysclassfiles\interface\tools 文件夹，可以看到有一个 i8086_installer.exe 程序，它是作者团队开发的一个 S86 汇编程序集成开发环境 i8086 IDE。包中涵盖了微软的 MASM 中的汇编器 ml.exe，链接器 link.exe，调试器 debug.exe，DOS 虚拟机 dosbox.exe，串口调试助手 UartAssist.exe，以及作者团队开发的 exe2coe、coe2txt 文件转换程序和一个全屏幕编辑器。

2.6.1　i8086 集成开发环境的使用

1. 准备工作

在安装 i8086 汇编集成开发环境前，需要确认以下安装环境是否满足：

- 使用 Windows 7 及以上版本操作系统（推荐使用 Windows 7，Windows 10）
- 准备安装的电脑上没有安装过 DOSBox

如果电脑上安装过了 DOSBox，请进行如下操作：

1） 请卸载之前安装过的DOSBox

2） 检查 C:\Users\“当前用户名”\AppData\Local\DOSBox 下是否存在 dosbox-0.74.conf文件，如果有，请将其删除

如图 2-9 所示，是目录示例，当前用户名称为“Michael”，所以目录如下：

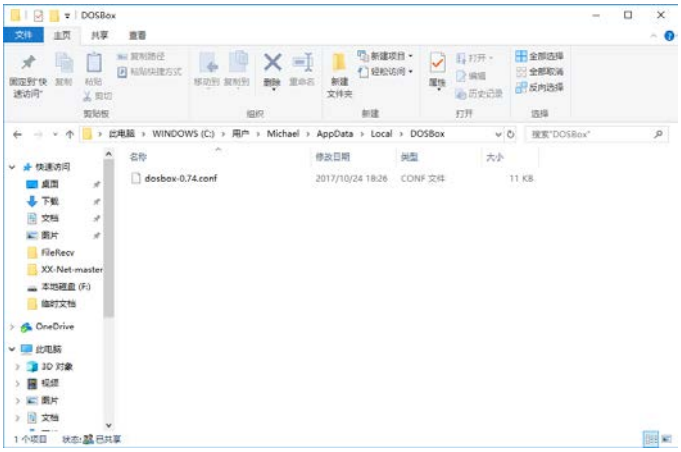


图 2-9 检查是否存在 dosbox-0.74.conf 文件

当前目录下存在 dosbox-0.74.conf，因此需要删除 dosbox-0.74.conf 文件

2. i8086 集成开发环境的安装

在 C:\sysclassfiles\interface\tools 目录下双击 i8086_installer.exe 文件，打开安装窗口，如

图 2-10 所示，按 Extract 按钮安装。

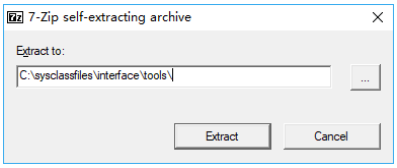


图 2-10 安装 i8086 集成环境

安装好后，会在 C:\sysclassfiles\interface\tools\i8086 目录下看到可执行文件 i8086.exe，如图 2-11 所示。

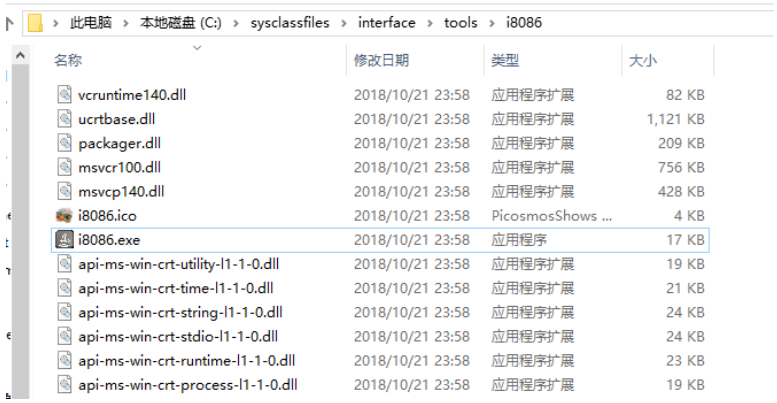


图 2-11 安装好 i8086h 后的目录

双击 i8086.exe 即可以运行 i8086 汇编集成开发环境，运行后界面如图 2-12 所示。

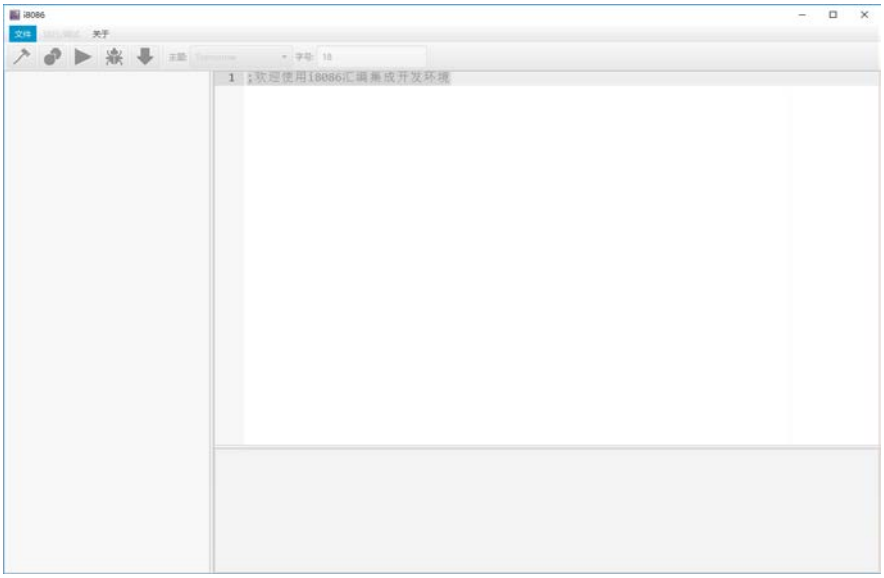


图 2-12 i8086 界面

i8086 在首次启动时，会先启动 DOSBox 保证之后 DOSBox 的启动正常。

注意：如果第一次启动 i8086 出现程序无法打开的错误，请尝试转到 C:\sysclassfiles\interface\tools\i8086\app\dosbox 下，手动双击 DOSBox.exe 执行该程序，停留数秒后，关闭 DOSBox.exe 的窗口，然后再执行 i8086.exe。另外，所有工程目录中请不要出现空格。

在进入界面后，默认为关闭工程的状态，所以界面上只有文件菜单下的新建工程、打开

工程、退出程序以及关于菜单是可用的，其它部分全部会被禁用。

3. i8086 集成开发环境主要功能描述

1） 新建项目

选择菜单栏上的文件 → 新建项目或者使用快捷键 Ctrl+N 来创建新项目。在新建项目过程中，重名会覆盖之前文件，并清空全部之前的生成文件。创建新项目需要指定新创建的项目的文件夹和项目中使用的文件名，之后会创建 ASM 文件。项目中全部生成文件都会被命名为与 ASM 文件名相同的名称。

2） 打开项目

选择菜单栏上的文件 → 打开项目或者使用快捷键 Ctrl+O 来打开已有项目。在打开项目过程中，会自动检查之前的同名生成文件。打开项目需要指定项目的文件夹和项目中使用的文件名，之后会打开 ASM 文件。项目中全部生成文件都会被命名为与 ASM 文件名相同的名称。

3） 保存项目

选择菜单栏上的文件 → 保存项目或者使用快捷键 Ctrl+S 来保存当前项目。保存项目会将全部的文件存留在预设的目录中，将文本编辑器中的内容存到 ASM 文件。

4） 关闭项目

选择菜单栏上的文件 → 关闭项目或者使用快捷键 Ctrl+E 来保存当前项目。关闭项目前将会询问是否保存项目。保存项目操作同上，不保存保持当前状态（ASM 不写入，新建项目保持空白）

5） 退出程序

选择菜单栏上的文件 → 退出程序或者使用快捷键 Alt+F4 来退出程序退出程序将不会询问是否保存项目，将直接退出程序，请一定在退出前保存好项目内容！

6） 汇编（Shift+F5，  ）

使用外部 MASM 的 ml.exe 以及 masm.exe 对源码进行汇编，生成 OBJ 文件和 LST 文件，可以到文件资源管理器中进行查看。

7） 汇编并连接（Shift+F6，  ）

使用外部 MASM 的 ml.exe 以及 masm.exe 对源码进行汇编，并使用 link.exe 连接生成 OBJ 文件、LST 文件和可执行程序 EXE，可以到文件资源管理器中进行查看。注意生成的可执行程序是 16 位可执行程序，并不兼容 Windows 7 及其以上版本，需要在 DOSBox 模拟器中运行，不可在 Windows 下双击执行。

8） 在DOSBox中运行（Shift+F7，  ）

会自动配置和运行 DOSBox，并启动生成的可执行程序进行仿真运行。**运行完后，请关闭 DOSBOX 窗口，**否则会阻塞 i8086 的继续执行。

9） 在DOSBox中调试（Shift+F8，  ）

会自动配置和运行 DOSBox，并启动调试器对生成的可执行程序进行调试。**调试完后，请关闭 DOSBOX 窗口，**否则会阻塞 i8086 的继续执行。

10）　生成下载文件（Shift+F10，  ）

该功能只有在做第四章和第五章实验的时候才会用到。此功能会将 exe 文件自动转换为两种格式的可下载到 Minisys 板的文件，一种是可提供给系统板进行内存下载的.COE 文件，另一个是可通过串口下载的.TXT 文件，可以到文件资源管理器中进行查看。.COE 文件必须利用 Vivado 打开 S86_SimpleSys.xpr 工程，用该文件的内容替换掉 S86_SimpleSys.xpr 工程中的 ini.coe 文件的内容，并重新生成 ram_bios IP 核，然后重新生成.bit 文件并下载，这个过程大概需要 10 分钟甚至更长时间。.TXT 文件只需要用 Vivado 软件将资源包中配的 C:\sysclassfiles\interface\assembler_lab\for Minisys\S86_Simple_System_Top.bit 下载到板子上，然后根据下面的步骤用串口加载.TXT 程序就可以了，这一过程大概只需要 30 秒不到的时间。

11）　其它功能描述

（1）　可以对文本编辑器的主题和文字大小进行修改

主题限定于下拉菜单中给定的常用主题，文字框只接受整数字号，其他情况将自动转换。

（2）　树状列表仅供对文件存在进行检查使用

当前版本并不支持双击打开文件的操作，只是检查文件列表而设立

（3）　底部终端显示器显示当前执行状态

在编译、连接以及生成内存文件时有效，其它状态下不进行输出。

2.6.2　DEBUG主要命令

DEBUG 是为汇编语言设计的一种调试工具，它通过单步和设置断点等方式为汇编语言程序员提供了非常有效的调试手段。

在 DOS 的提示符下，可输入命令：

```
C>DEBUG  [d：][path][filename[.ext]][parml][parm2]
```

其中，文件名是被调试文件的名字。如用户输入文件名，则 DEBUG 将指定的文件装入存储器中，用户可对其进行调试。如果未输入文件名，则用户可以用当前存储器的内容工作，或者用 DEBUG 命令 N 和 L 把需要的文件装入存储器后再进行调试。命令中的 d 指定驱动器，Path 为路径，parml 和 parm2 则为运行被调试文件时所需要的命令参数。

在 DEBUG 程序调入后，将出现提示符“-”，此时就可用 DEBUG 命令来调试程序了。下面介绍 DEBUG 的主要命令。

1. 显示存储单元的命令D（DUMP）

格式为：-D[address]，或-D[range]

例如，按指定范围显示存储单元内容的方法为：

```
-D100 11F
1636：0100  00 8B 36 EB D8 8B 0E E9-D8 8B D6 E3 42 51 56
5B   ..6.........BQV[
1636：0110  2B DE 59 03 CB 8B D6 C6-06 CD DC 00 34 00 25 16
+.Y.........4.%.
```

其中，0100H 至 011FH 是 DEBUG 显示的单元内容。左边用十六进制数表示每个字节，右边用 ASCII 字符表示每个字节，表示为不可显示的字符。这里没有指定段地址，D 命令自动显示 DS 段的内容。如果只指定首地址，则显示从首地址开始的 128 个字节的内容。如果完

全没有指定地址，则显示上一个 D 命令显示的最后一个单元后的内容。以上命令也可写为

```
-D100L20
```

其中 L 表示长度。

注意：在 DEBUG 中，与汇编语言不同，所有数默认为十六进制数。

2. 修改存储单元内容的命令E（Enter）

输入命令 E 有两种格式。

1） 用给定的内容表来替代指定范围的存储单元内容。命令格式为：

```
-E  address  [list]
```

例如，–EDS:100 F3'XYZ'8D

其中 F3，'X'，'Y'，'Z'和 8D 各占一个字节，该命令可以用这五个字节来替代存储单元 DS:0100H 到 0104H 的原先的内容。

2） 采用逐个单元相继修改的方法。命令格式为：

```
-E  address
```

例如，-ECS:100，则可能显示为：

```
14A0：0100  00.
```

如果需要把该单元的内容修改为 78，则用户可以直接输入 78，再按“空格”键可接着显示下一个单元的内容，如下：

```
14A0：0100  00.78   8B.
```

这样，用户可以不断修改相继单元的内容，直到用 Enter 键结束该命令为止。

3. 检查和修改寄存器内容的命令R（Register）

它有三种格式。

1） 显示CPU内所有寄存器内容和标志位状态

其格式为：-R。例如：

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000
DI=0000
DS=14A0  ES=14A0  SS=14A0  CS=14A0  IP=0100   NV UP EI PL NZ NA
PO NC
14A0：0100 F3            REPZ
14A0：0101 7879          JS      017C
```

2） 显示和修改某个寄存器内容

其格式为：-R　register。例如，输入：-rAX，系统响应如下：

```
AX 0000
:
```

即 AX 寄存器的当前内容为 0000H，如不修改则按 Enter 键，否则，可输入欲修改的内容，如：

```
-rbx
BX 0000
```

：369

则把 BX 寄存器的内容修改为 0369H。

3） 显示和修改标志位状态

命令格式为：-RF。系统响应如下：

```
NV UP EI PL NZ NA PO NC  -
```

此时，如不修改其内容可按 Enter 键，否则，可输入欲修改的内容。如：

```
NV UP EI NG NZ NA PO NC  -ngovdnpo
```

即可，可见输入的顺序可以是任意的。但必须按照标志位的置/复位表示方法书写。

4. 运行命令G（Go）

其格式为：-G[=address1][address2][address3...]]

其中，地址 l 指定了运行的起始地址，如不指定则从当前的 CS:IP 开始运行。后面的地址均为断点地址，当指令执行到断点时，就停止执行并显示当前所有寄存器及标志位的内容和下一条将要执行的指令。

5. 跟踪命令T（Trace）

它有两种格式。

1） 逐条指令跟踪

其格式为：-T[=address]

从指定地址起执行一条指令后停下来，显示所有寄存器内容及标志位的值。如未指定地址，则从当前的 CS:IP 开始执行。

2） 多条指令跟踪

其格式为：-T[=address][value]

从指定地址起执行 *n* 条指令后停下来，*n* 由 value 指定。

6. 汇编命令A（Assemble）

其格式为：-A　[address]

该命令允许输入汇编语言语句，并能把它们汇编成机器代码，相继存放在从指定地址开始的存储区中。

7. 反汇编命令U（Unassemble）

1） 从指定地址开始，反汇编 32 个字节

其格式为：-u[address]。例如：-U100。

如果地址被省略，则从上一个 u 命令的最后一条指令的下一个单元开始显示 32 个字节。

2） 对指定范围内的存储单元进行反汇编

格式为：-u[range]。例如：-U100 10C 或 U100LD。

8. 命名命令N（Name）

其格式为：-N filespecs [filespecs]

该命令把两个文件标识符格式化在 CS：5CH 和 CS：6CH 的两个文件控制块中，以便在其

后用 L 或 W 命令把文件装入或存盘。filespecs 的格式可以是：[d：][path]filename[.ext]。例如：

```
-N myprogL
-L
```

可把文件 myprogL 装入存储器。

9. 装入命令L（Load）

1） 把磁盘上指定扇区范围的内容装入到存储器从指定地址开始的区域中

其格式为：

```
-L[address][drive sector number]
```

2） 装入指定文件，其格式为：-L[address]。

此命令装入已在 CS：5CH 中格式化了的文件控制块所指定的文件。如未指定地址，则装入 CS：0000H 开始的存储区中。

10. 写命令W（write）

1） 把数据写入磁盘的指定扇区

其格式为：-W address drive sector number。

2） 把数据写入指定的文件中

其格式为：-W[address]。

此命令把指定的存储区中的数据写入由 CS：5CH 处的文件控制块所指定的文件中。如未指定地址则数据从 CS：0100H 开始。要写入文件的字节数应先放入 BX 和 CX 中。CX 中为低 16 位，若数据长度大于 64 KB，则使用 BX 记录高 16 位。

11. 帮助命令

其格式为：-?

显示 DEBUG 所有命令简明列表。

12. 退出DEBUG命令Q（Quit）

其格式为：-Q

它退出 DEBUG，返回 DOS。本命令并无存盘功能，如需存盘应先使用 W 命令。

第 3 章 8086 汇编实验（For PC）

由于 S86 在汇编一级与 8086 兼容，在后续接口实验中会用到汇编程序设计，因此，本章中专门开设 8086 实验，本章的实验需在 PC 上完成。需要的工具是 MASM5.0 及以上或 ml，LINK。如果在 Windows7 及以上操作系统，还需要 DOSBOX0.7 的支持。以上软件均被集成到 i8086 IDE 中。

3.1　汇编语言程序上机过程

3.1.1　Sample程序的汇编、链接与调试

1. 实验目的

学会将汇编语言源程序录入进计算机、利用 i8086 IDE 集成的 ml.exe 进行汇编，LINK 进行链接，并用 DEBUG 调试 16 位程序的全部过程。本实验大家不必了解程序细节，只是为了熟悉开发环境和上机过程。

2. 实验内容

1) 按照第 2 章安装好 i8086 开发环境。
2) 熟悉 16 位汇编程序的上机过程和调试过程。

3. 实验预习

1) 认真复习源程序编写和汇编、链接的操作步骤。
2) 认真复习 DEBUG 的各种命令。

4. 实验步骤

1） 打开i8086 IDE

在 C:\sysclassfiles\interface\tools\i8086 目录下双击 i8086.exe，得到如图 2-12 所示的界面。

2） 源程序的输入

选择菜单栏上的文件 → 新建项目或者使用快捷键 Ctrl+N 在 C:\sysclassfiles\interface\assembler_lab\for PC 中创建新项目，项目名称为 Ex3_1_1.asm。输入下面的程序并保存，如所示。

```
DATA SEGMENT
  BUFF   DW  3C6DH
  STRING DB  'PRINT HEX.',0AH,0DH,'$'
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS : DATA
START:  MOV  AX,DATA
```

```
        MOV  DS,AX
        MAV  DX,OFFSET  STRING
        MOV  AH,09H
        INT  21H
        LEA  SI,BUFF
        MOV  BX,[SI]
        MOV  CH,4
A1:     MOV  CL,4
        ROL  BX,CL
        MOV  AL,BL
        AND  AL,0FH
        ADD  AL,30H
        CMP  AL,3AH
        JL   A2
        ADD  AL,7H
A2:     MOV  DL,AL
        MOV  AH,2
        INT  21H
        DEC  CH
        JNZ  A1
        MOV  AH, 4CH
        INT  21H
CODE ENDS
END START
```

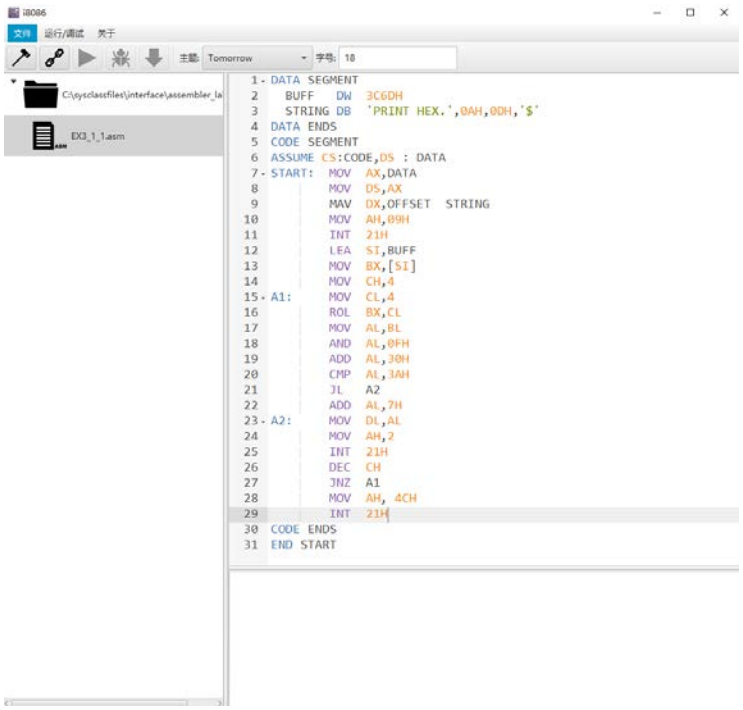


图 3-1 输入程序

3） 汇编源程序

点击按钮或按 Shift+F5 键汇编上述输入的程序，此时出现所示的图 3-2 所示画面。

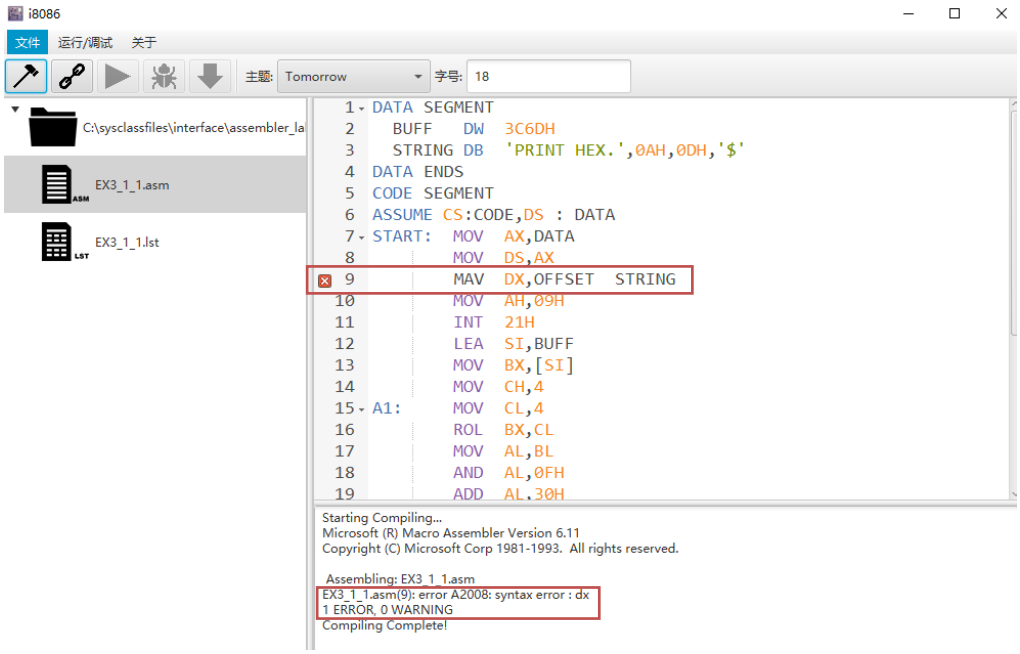


图 3-2 汇编出错

图 3-2 指出源程序有一处错误，该错误在第 9 行，仔细检查，发现第九行错吧 MOV 写成了 MAV，将这个错误改正过来后，再次点击，就可以看到图 3-3 所示的界面，说明汇编成功。从 i8086 左边的窗口可以看到生成了 EX3_1_1.lst 和 EX3_1_1.obj 两个文件。

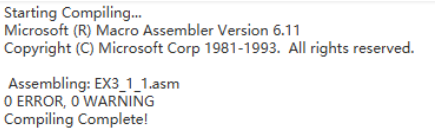


图 3-3 链接 obj 传文件

4） 汇编与链接

点击按钮或按 Shift+F6 键链接刚生成的 EX3_1_1.obj 文件，出现如图 3-4 所示的界面说明完成链接过程。上述过程顺利完成时，会生产 EX3_1_1.exe 文件。

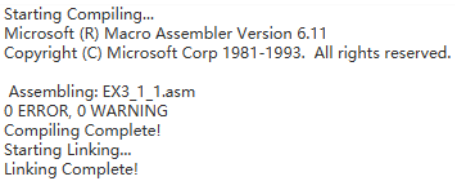


图 3-4 链接成功

5） 调试

点击按钮或按 Shift+F8 键启动 DosBox 并执行 DEBUG　SE3_1_1.EXE 命令将该程序调入到 DEBUG 环境中调试。

1） 用 U 命令察看源代码

2） 用 D 命令察看数据段的数据

3） 用 R 命令察看寄存器状态

4） 用 T 命令单步跟踪程序执行

5） 用 G 命令设置断点，调试软件中断指令（不要用 T）

3.1.2　字符串的输入与输出

1. 实验目的

1) 进一步熟悉汇编上机过程和调试过程
2) 熟悉人机交互程序设计

2. 实验内容

编完整程序，利用 DOS 系统功能调用，从键盘输入一个字符串，并将该字符串从屏幕上换行后输出，用 DEBUG 调试该程序。

3. 实验预习要求

复习人机交互的设计方法。

4. 实验步骤

同 3.1.1，程序自行编写。

3.2　顺序程序设计实验

3.2.1　四则运算

1. 实验要求

通过这一部分的实验，进一步熟悉汇编过程和 DEBUG 调试过程；掌握用汇编语言编写顺序程序的方法。

2. 实验内容

x，y，z，v 均为 16 位带符号数，计算(v-(x*y+z-540))/x

3. 实验预习

弄懂本实验中提供的源程序。

4. 实验步骤

1） 输入下面的源程序。

```
DATA SEGMENT
X  DW ?
       Y   DW ?
       Z   DW ?
       V   DW ?
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS:CODE, DS : DATA
START:MOV    AX, DATA
      MOV    DS, AX
MOV       AX, X
          IMUL      Y                  ;  x*y
          MOV       CX, AX
          MOV       BX, DX
          MOV       AX, Z
          CWD
          ADD       CX, AX
          ADC       BX, DX             ; x*y+z
          SUB       CX, 540
          SBB       BX, 0             ;x*y+z-540
          MOV    AX, V
          CWD
          SUB     AX, CX
          SBB     DX, BX               ; v-(x*y+z-540)
          IDIV    X                    ; (v-(x*y+z-540))/x
CODE ENDS
        END START
```

2） 汇编、链接、调试程序

3.2.2　查表的平方值

1. 实验目的

通过这一部分的实验，进一步熟悉汇编过程和 DEBUG 调试过程；掌握用汇编语言编写顺序程序的方法和查表（换码指令）的用法。

2. 实验内容

在内存中从 Table 开始的 10 个单元中连续存放 0 到 9 的平方值，任给一个 0 到 9 的数 X，该数存放在内存单元 XX 中，查表求 X 的平方值，并将结果存于内存 YY 单元中。编写程序，并在 DEBUG 中进行调试和验证结果。（提示：考虑平方表的每一项需要什么数据类型才合适， XLAT 指令是否合适？应该如何查表？）

3. 实验步骤

同 3.1.1，程序自行编写。

3.2.3　非压缩BCD码转二进制

1. 实验目的

通过这一部分的实验，进一步熟悉汇编过程和 DEBUG 调试过程；掌握用汇编语言编写顺序程序的方法和加减乘除指令的用法。

2. 实验内容

假设 CX:BX 中放了 4 位非压缩的 BCD 码表示的十进制数 4386，请编写完整程序将这个数转成 2 进制数放到 DI 寄存器中，并用 DEBUG 调试和验证之。

3. 实验预习

1) 弄清楚压缩 BCD 码和非压缩 BCD 码的概念
2) 考虑好算法流程

4. 实验步骤

同 3.1.1，程序自行编写。

3.3 分支程序设计实验

3.3.1 简单分支程序设计

1. 实验目的

通过本实验，熟练运算类指令对标志位状态的影响，以及标志位状态的表示方法；掌握无条件转移、条件转移指令的使用方法；掌握分支程序设计和调试方法。

2. 实验内容

如果 X>50，则显示 TOO_HIGH；否则计算 X-Y，溢出显示 OVERFLOW，若无溢出则 |X-Y|→RESULT。

3. 实验预习

1) 复习分支程序设计的相关指令，给出算法流程图。
2) 复习运算类指令
3) 弄懂本实验提供的源程序

4. 实验步骤

1） 输入下面的源程序

```
DATA SEGMENT
   X   DB ?
   Y   DB ?
   STR1   DB 'TOO_HIGH!',0DH,0AH,'$'
   STR2   DB 'OVERFLOW!',0DH,0AH,'$'
   RESULT DB ?
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS : DATA
START:    MOV   AX,DATA
```

```
            MOV   DS,AX
MOV    AL, X
CMP    AL, 50
JG     TOO_HIGH
SUB    AL, Y
JO     OVERFLOW
JNS    NONNEG
NEG    AL      ;计算绝对值
 NONNEG:  MOV    RESULT, AL
          JMP    EXIT
TOO_HIGH:  LEA   DX,STR1          ;显示'TOO_HIGH'
          JMP    DISPLAY
OVERFLOW: LEA       DX,STR2           ;显示'OVERFLOW'
DISPLAY:  MOV AH,09H
          INT    21H
EXIT:     MOV AH,4CH
          INT    21H
CODE ENDS
         END START
```

2） 汇编、链接、调试程序

3.3.2　数的十进制输出

1. 实验目的

通过本实验，进一步掌握无条件转移、条件转移指令的使用方法；雪狐用十进制形式在屏幕上输出一个数。

2. 实验内容

已知数据段有以下定义：

```
NUM    DW   3570
STRING    DB 5 DUP(20H),'$'  ; 20H 为空格的 ASCII 码
```

请编写完整程序，在屏幕上以十进制的形式将 NUM 这个数打印出来，可以借助 STRING 这个字符串。（NUM 这个数可以定义为一个任意字型数）。

3. 实验预习

复习分支程序设计的相关指令，给出算法流程图。

4. 实验原理

二进制数转为 10 进制数采用将数除以 10 取余数，上继续除以 10 取余数，直到商为 0. 需要注意的是，这样的方法先得到的是个位数，然后是 10 位数，百位数，……。所以需要一个 STRING 字符串从右向左存放各位数字的 ASCII 码。

5. 实验步骤

同 3.1.1，程序自行编写。

3.4　循环程序设计

3.4.1　在字符串中替换字符

1. 实验目的

通过实验，可以掌握循环结构的各种实现方法，进一步了解循环结构中初始化部分、循环体部分、循环控制部分的功能以及他们彼此之间的关系。

2. 实验内容

已知字符串 string 包含有 256BYTE 的内容，将其中所有的'$'符号替换成空格（''）

3. 实验预习

1) 认真复习循环程序设计方法
2) 给出算法流程
3) 弄懂本实验提供的源程序

4. 实验步骤

1） 输入下面的源程序

```
D SEGMENT
  BUF DB 56 dup ('1'), 100 dup ('$'), 100 dup ('a')
D ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS : DATA
MAIN    PROC FAR
      PUSH DS
      MOV AX,0
      PUSH AX
      MOV AX,DATA
         MOV DS,AX
      MOV CX,256
      LEA SI,BUF
A30:  CMP BYTE PTR [SI],'$'
      JE CHANGE
      JMP A31
CHANGE: MOV BYTE PTR [SI],' '  ; ' '空格的 ASCII 码是 20H
A31:  INC SI
      LOOP A30
      RET
MAIN    ENDP
```

```
CODE ENDS
        END MAIN
```

2） 汇编、链接、调试程序

3.4.2　统计字符串中字符个数

1. 实验目的

通过实验，进一步掌握循环结构的各种实现方法，了解循环结构中初始化部分、循环体部分、循环控制部分的功能以及他们彼此之间的关系。

2. 实验内容

已知数据段有以下定义：

```
STRING BYTE 'I have a dream that I can make a CPU by myself.',0DH,0AH,'$'
LEN    WORD ?
NUM    BYTE 5 DUP(20H),'$'      ; 20H 为空格的 ASCII 码
```

请编写完整 16 位汇编程序，统计以‘$’字符结束的字符串 STRING 的字符个数（不算'$'），将个数放入 LEN 中，并在屏幕上以十进制的形式将 LEN 打印出来，可以借助 NUM 这个字符串。

3. 实验预习

1) 复习循环程序设计的有关知识
2) 复习十进制数屏幕输出的犯法

4. 实验步骤

同 3.1.1，程序自行编写。

3.5　子程序设计

3.5.1　十进制到十六进制转换

1. 实验目的

通过本实验，掌握子程序的定义和调用方法。通过程序调试，进一步理解 CALL 指令和 RET 指令的功能，

2. 实验内容

从键盘取得一个十进制数，将其以十六进数形式显示出来。要求设计 3 个子程序，第一个 DECIBIN，该子程序从键盘输入一个小于 65536 的十进制数，放在 BX 中；第二个子程序 BINIEX 将 BX 寄存器种的数用十六进制形式输出到屏幕上;第三个子程序输出回车换行。

3. 实验预习

仔细考虑前两个子程序的算法流程。

4. 实验步骤

同 3.1.1，程序自行编写。

3.5.2　计算N!

1. 实验目的

通过本实验，进一步掌握子程序的定义和调用方法，掌握子程序调用时参数传递的方法。

2. 实验内容

利用递归程序，计算 N!。具体要求：用键盘输入一个数 N(1~6 之间)，利用一个递归过程 FAC 来计算 N!（N 放在 AL 中，结果在 DX 中），然后将计算的结果以十进制形式打印到屏幕上。

3. 实验预习

1) 认真复习子程序设计的方法
2) 认真领会递归程序设计的方法

4. 实验步骤

同 3.1.1，程序自行编写。

3.6　综合实验

3.6.1　判断回文

1. 实验目的

进一步加强对汇编程序设计的理解，加强字符串输入，循环、分支程序设计和字符串的输出。

2. 实验内容

所谓回文字符串是指一个字符串正读和倒读都是一样的，例如字符串‘ABCDEFFEDCBA’就是一个回文字符串，而字符串‘ABCFDDCAB’就不是回文字符串。现在编写完整的 16 位汇编程序，输入一个字符串，判断该字符串是否为回文字符串，并用“It is a palindrome”或“It is NOT a palindrome”作为输出。

3. 实验步骤

同 3.1.1，程序自行编写。

3.6.2　有序数组中找数

1. 实验目的

学会在有序数组中找到一个数，也学会在有序数组中删除一个数。

2. 实验内容

请编写 16 位完整汇编程序，在一个升序字节数组 BUFF 中查找数 N，找到后将此数从数组中删除，并使得 CF=0；没找到返回 CF＝1。数组的首地址和末地址为 A_HEAD 和 A_END。

3. 实验步骤

同 3.1.1，程序自行编写。

3.6.3　带反馈输入十进制数字的二进制数出

1. 实验目的

学会十进制数的二进制输出。综合汇编的多种知识。

2. 实验内容

请编写完整程序从键盘读取字符，如果是十进制的‘0’～‘9’则在屏幕上输出该数的 8 位二进制码，并将数字依次存放到 BUF 开头的数组中，如果读入的字符是’Q’或者‘q’，则程序退出，其他情况在屏幕上打印“You must input 0~9, or ‘q’ or ‘Q’”。（如输入的字符是‘9’，则输出“00001001”）.提示：输出一个数的 2 进制形式应该从最高位开始输出，可以将此段程序定义成一个过程。

3. 实验步骤

同 3.1.1，程序自行编写。

第 4 章 S86 汇编实验（For Minisys）

本章针对 Minisys 实验板开设了 S86 汇编实验，目的是让学生能够体会在不同于 PC 的平台下如何编写汇编实验，尤其是在嵌入式环境下汇编程序的编写。

4.1　预备知识

4.1.1　使用S86_SimpleSys

为了在 Minisys 板上进行汇编实验，教学资源中提供了一个 S86_SimpleSys，该系统包括了 S86 处理器，存储器，串口下载部件、16 位 LED、16 位拨码开关、8 位数码管和 4×4 键盘等外设，可以进行简单的汇编应用程序的开发，并提供了这些外设的系统功能调用。但由于 S86 没有调试功能，因此调试程序还得在 PC 平台上使用诸如 DEBUG 这样的调试手段，但无法调试 S86 的系统功能调用。

S86_SimpleSys 存放在学生资源包 C:\sysclassfiles\interface\assembler_lab\for Minisys\中。

1. S86_SimpleSys汇编实验常规步骤

1） 汇编程序编写

按照 4.1.3 中的汇编程序模板，参看 3.1.1 节，在 i8086 中编写汇编程序。

2） 编译链接

参看 3.1.1 节，在 i8086 中编译、链接、生成相应的 EXE 文件。

3） 将EXE文件转换成COE文件和TXT文件

参看 2.6.1 节，在 i8086 中将 exe 文件转换成一个 XXX.coe 文件和 XXX.txt 文件。.coe 文件和.txt 文件分别采用不同的方式加载到 Minisys 板子上。作为汇编程序设计，建议大家采用.txt 文件下载，因为这种方法不需要重新生成.bit 文件，下载速度快。所以本章只介绍 TXT 文件的加载。

4） 生成比特流文件和下载比特流文件

用 USB 下载线将 Minisys 板的 PROG UART（2016 版的 Minisys 板）或 Type C（2017 版的 Minisys 板）与 PC 机的 USB 相连，**打开 Minisys 板的电源**。打开 Vivado 软件，点击 Flow 下的 Open Hardware Manager，如图 4-1 所示。

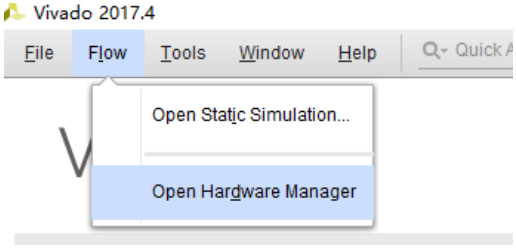


图 4-1 打开硬件管理器

出现图 4-2 所示的界面。

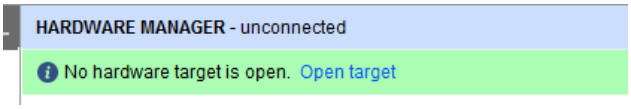


图 4-2 Hardware Manager

点击 Open target。在弹出的窗口中点击 Auto Connect。此时出现图 4-3 所示的滚动条。

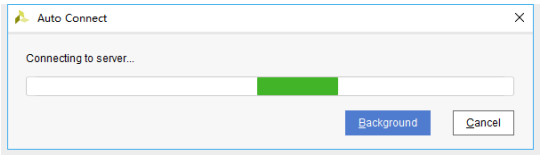


图 4-3 寻找硬件

硬件连接上后，会出现图 4-4 所示界面。

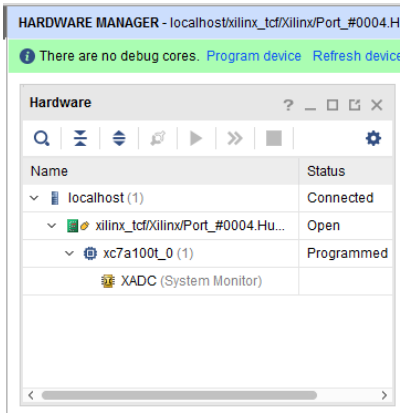


图 4-4 找到硬件

点击 Program devices。出现的 Program Device 窗口（图 4-5）。

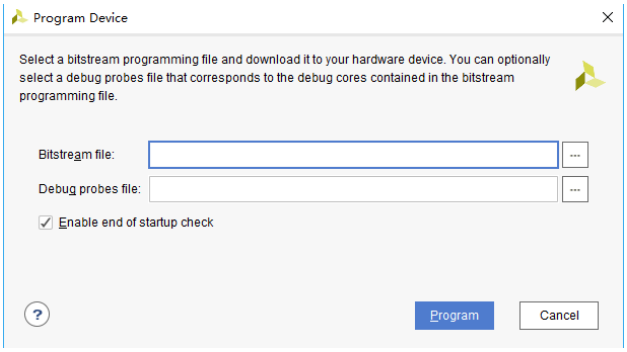


图 4-5 Program Device 窗口

在该窗口的 Bitstream file:中填入 C:\sysclassfiles\interface\assembler_lab\for Minisys\S86_Simple_System_Top.bit。也可以点击边上的…，在资源管理器中选择上述文件，如图 4-6 所示。点击 **Program**。

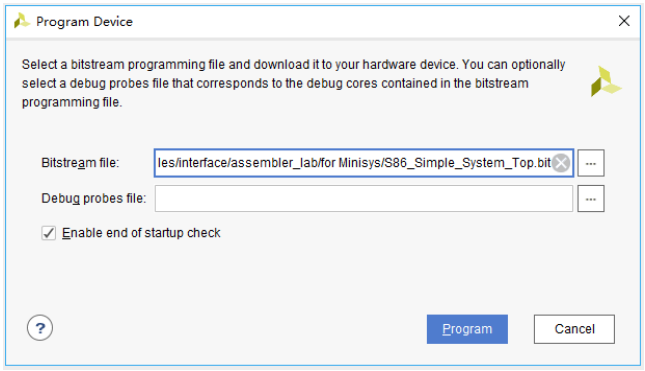


图 4-6 下载比特流文件

出现如图 4-7 所示的正在下载的进度条。

图 4-7 下载进度条

比特流文件下载成功后，按一下 Minisys 板上的 FPGA 复位按键，你会看到把个数码管同时循环显示 11111111，22222222，……，FFFFFFFF 字样，YLED7 到 GLED0 这 16 个 LED 灯反复做闭幕与开幕动作，表明 S86 已经正常运行。

5） 加载.txt文件

.TXT 文件生成并且 S86 正常运行后，就可以将.txt 文件通过串口加载到 Minisys 板上的 S86 中运行了。具体步骤如下。

首先打开 Windows 设备管理器，查看开发板使用的串口号，如图 4-8 所示，串口的端口号为 COM6。

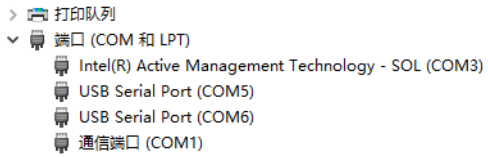


图 4-8 查看端口号

在 C:\sysclassfiles\interface\tools\i8086\app\upg 文件夹中双击 UartAssist.exe，打开串口工具，选择上面所显示的端口，并按所示设置。

图 4-9 串口设置

点击打开按钮打开该串口，发送区设置为按十六进制发送，并启用文件数据源，选择待

发送文件，如图 4-10 所示。

图 4-10 选择要发送的文件

先按下开发板上的 S3 按钮（如图 4-11 所示），让 S86 进入下载状态。

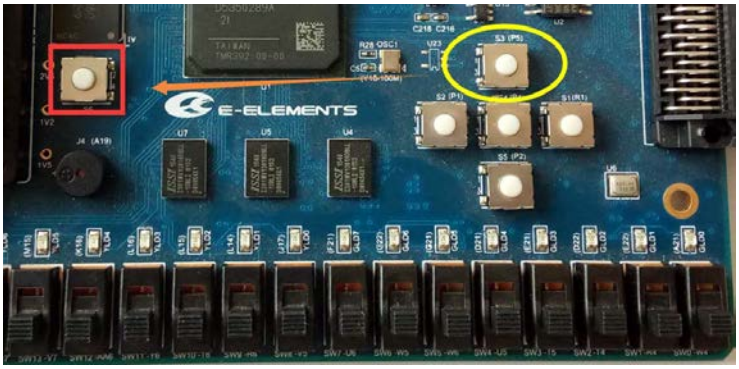


图 4-11 右边是 S3 按钮，左边是 Reset 按钮

点击串口工具中的发送按钮，串口工具左下角显示“正在发送数据...”，等待 15s 左右。串口工具左下角再次显示“就绪”时，代表加载完成。

加载完成后，按下开发板的 FPGA 复位按按钮（图 4-11 中左侧的按钮），CPU 开始运行所加载的程序。

4.1.2　S86_SimpleSys系统功能调用

在 S86_SimpleSys 中进行汇编实验，系统提供了相应的系统功能调用来满足汇编实验中简单的输入输出工作。

1. 系统功能调用的一般格式

格式：`INT　N`

系统功能调用采用中断指令方式，其中 N 为中断号。AH 中需要放功能号，根据需要还

需要在 DX 或 AL 中放入调用参数。

2. 系统功能调用详解

表 4-1 是 S86_SimpleSys 提供的系统功能调用，利用这些系统功能调用，汇编程序可以使用 Minisys 实验板上的 16 位 LED、16 位拨码开关、8 位数码管和 4×4 键盘

表 4-1 系 S86_SimpleSys 统调用说明

中断号 N	功能号	调用参数	返回参数	功能
30H	AH=0	DX		将 dx 内容输出到 YLED7~YLED0,GLED7~GLED0
31H	AH=0		DX	获取 SW16~SW0 的值保存到 dx 中
32H	AH=0	AL		设置某个数码管是否允许显示（寄存器的 d7~d0 位对应 A7~A0 这 8 个数码管），1 为显示，0 为不显示（不亮）
	AH=1	DX		将 dx 中的数据 4 位为一组，从左到右分别以 16 进制形式显示在 A3~A0 的数码管上
	AH=2	DX		将 dx 中的数据 4 位为一组，从左到右分别以 16 进制形式显示在 A7~A4 的数码管上
33H	AH=0		AL	读取键盘值保存到 al 中（D4 为状态位，等于 1 表示有键，此时 D3~D0 为键值；D4 等于 0 表示无键，D3~D0 无效）

比如：如果我们需要在数码管上显示时、分、秒，则我们只需要 8 位数码管的低 6 位显示，高两位时钟为暗的（不显示），我们可以用以下系统功能调用来实现。

```
MOV   AL,00111111B
MOV   AH,0
INT   32H
```

4.1.3　汇编编程模板

由于 S86 是逻辑，没有操作系统，因此程序没有退出并返回操作系统的步骤，在 S86 上运行的程序都必须是一个大的死循环。另外，**在 Minisys 板子的 S86 系统中运行的程序，其源程序不能用简单段定义法。也就是说本章实验和第 5 章实验的汇编程序都必须用完整段定义法。**因此本书给出下面的 S86 汇编程序模板供大家参考。

1. 没有数据段的程序模板

```
CODE SEGMENT ‘CODE’
   ASSUME CS:CODE
START:
   ;用户代码
   JMP START
CODE  ENDS
      END START
```

2. 有数据段的程序模板

```
DATA SEGMENT 'DATA'
    ;用户数据
DATA ENDS
```

```
CODE SEGMENT 'CODE'
   ASSUME CS:CODE, DS: DATA
START:
   MOV AX, 0080H  ;数据段从内存的 00800H 开始，因此 DS=0080H
   MOV DS, AX
   ;用户代码
   JMP START
CODE  ENDS
      END START
```

4.2　顺序程序设计

4.2.1　读取拨码开关的数据输出到LED

1. 实验目的

1) 熟练掌握 S86_SimpleSys 汇编程序设计、编译、链接、下载执行的方法。
2) 学会利用系统功能调用进行拨码开关输入与 LED 输出。

2. 实验内容

利用程序模版完善带???的程序段实现以下功能：从拨码开关（SW15~SW0）读取数据输出到 LED（YLED7~YLED0，GLED7~GLED0）上。

```
CODE SEGMENT
ASSUME CS:CODE
START:
      ???
      JMP START
CODE ENDS
      END START
```

3. 实验预习

认真阅读 4.1 的预备知识，掌握 S86_SimpleSys 的系统功能调用。

4. 实验步骤

参见 4.1.1 的实验步骤。

4.2.2　两数相加

1. 实验目的

1) 进一步熟悉 S86_SimpleSys 汇编程序设计、编译、链接、下载执行的方法。
2) 学会数码管的输出
3) 练习加法指令，熟悉顺序程序设计

2. 实验内容

利用程序模版完善带???的程序段实现以下功能：从拨码开关输入两个 8 位二进制数（A、B），将这两个数的和以 16 进制数形式输出到数码管上。A 等于 SW15~SW8 的值；B 等于 SW7~SW0 的值。

```
CODE SEGMENT
ASSUME CS:CODE
START:
???
      JMP START
CODE ENDS
      END START
```

3. 实验预习

1) 预习数码管输出的方法
2) 认识二进制数和十六进制数之间的关系
3) 复习加法指令

4. 实验步骤

参见 4.1.1 的实验步骤。

5. 思考与研讨

考虑减法、乘法的运算。如果数码管不能显示负数，且最多只有 8 位十六进制数，则减法和乘法运算有什么样的限制？

4.2.3　查表求平方并以十六进制输出

1. 实验目的

1) 熟悉查表（换码）指令的用法

2. 实验内容

利用程序模版完善带???的程序段实现以下功能：在内存中从 Table 开始的 10 个单元中连续存放 0 到 9 的平方值，从拨码开关中以二进制形式输入一个 0 到 9 的数 X，查表求 X 的平方值，以 16 进制形式输出到数码管上。

```
DATA SEGMENT
    TABLE DB ???
DATA  ENDS

CODE SEGMENT
   ASSUME CS:CODE, DS: DATA
START:
   MOV AX, 0080H  ;数据段从内存的 0080 段开始
```

```
    MOV DS, AX
    ???
    JMP START
CODE  ENDS
      END START
```

3. 实验预习

预习查表（换码）指令的用法。

4. 实验步骤

参见 4.1.1 的实验步骤。

5. 思考与研讨

如果是 20 以内的数的平方运算用查表法还能不能用换码指令？如果不能，怎么修改程序？

4.3　分支与循环程序设计

4.3.1　二进制输入，十进制输出

1. 实验目的

1) 熟悉分支语句的使用和分支程序设计。
2) 学会循环程序设计
3) 学会将 2 进制数转为 10 进制数输出。

2. 实验内容

利用程序模版实现以下功能：从拨码开关（SW12~SW0）输入一个 13 位无符号二进制数，以十进制的方式输出到数码管（A3~A0）上。

3. 实验预习

给出二进制数到十进制数转换的流程图。

4. 实验原理

二进制数转为 10 进制数采用将数除以 10 取余数，上继续除以 10 取余数，直到商为 0. 需要注意的是，这样的方法先得到的是个位数，然后是 10 位数，百位数，……。

5. 实验步骤

参见 4.1.1 的实验步骤。

6. 思考与研讨

16 位无符号 2 进制数转成十进制数在数码管上输出，应该如何修改程序？注意数码管

的输出。

4.3.2　十进制数的输入与输出

1. 实验目的

学会键盘的输入。

2. 实验内容

从 4×4 键盘中输入一个不大于 10000 的十进制的数（大于则数码管清 0），将该数以 10 进制形式输出到数码管（A3~A0）上。

3. 实验预习

考虑如何完成键盘输入。

4. 实验步骤

参见 4.1.1 的实验步骤。

5. 思考与研讨

如何将实验内容改为输出数据后，等待任意一个键，数码管清 0，并开始输入下一个数。

4.3.3　两个数的加减乘

1. 实验目的

1) 进一步加强分支程序和循环程序设计。
2) 学会带有菜单功能的程序设计。
3) 巩固加法、减法和乘法运算程序的设计。

2. 实验内容

从拨码开关输入两个 8 位二进制数（A、B），通过 4×4 键盘分别输入 1（加法）、2（减法）或 3（乘法）对这两个数进行运算，将计算结果以 10 进制的形式输出到数码管（A3~A0）上。A 等于 SW15~SW8 的值；B 等于 SW7~SW0 的值。（注意：如果减法计算结果为负数或者计算结果大于 9999 则显示 E）。程序循环往复。

3. 实验预习

1) 考虑程序应该如何做到循环往复。
2) 考虑如何让读键程序部分无须等待键盘输入。

4. 实验步骤

参见 4.1.1 的实验步骤。

4.4　子程序设计

4.4.1　判断回文

1. 实验目的

1) 熟悉子程序的编写。
2) 进一步熟悉分支程序、循环程序的编写。

2. 实验内容

从 4×4 键盘上输入由 8 个 0~9 的数字组成的字符串，在数码管上回显，并编写子程序判断该字符串是否为回文，如果是回文则 GLD7 亮，否则 RLD7 亮，按 0~9 任意一键进入下一轮查询。所谓回文字符串是指一个字符串正读和倒读都是一样的，例如字符串‘ABCDEFFEDCBA’就是一个回文字符串，而字符串‘ABCFDDCAB’就不是回文字符串。

3. 实验预习

给出判断回文的算法流程。

4. 实验步骤

参见 4.1.1 的实验步骤。

5. 思考与研讨

如果输入的是 2~8 个字符数不等的字符串，该如何判断回文？

4.4.2　利用递归程序，计算N!。

1. 实验目的

1) 巩固子程序设计以及参数传递。
2) 学会递归程序设计。

2. 实验内容

用 4×4 键盘每次输入一个数 N(1~6 之间)，则上利用一个递归过程 FAC 来计算 N!，然后将计算的结果以 10 进制的形式输出到数码管上。

3. 实验预习

给出递归算法的思路和流程。

4. 实验步骤

参见 4.1.1 的实验步骤。

4.5　综合实验

4.5.1　四则运算计算器

1. 实验目的

综合上面实验的训练结果，进一步加强汇编编程能力。

2. 实验内容

编写汇编程序完成简单的四则运算计算器功能：

1）计算器键盘布局如图 4-12 所示：

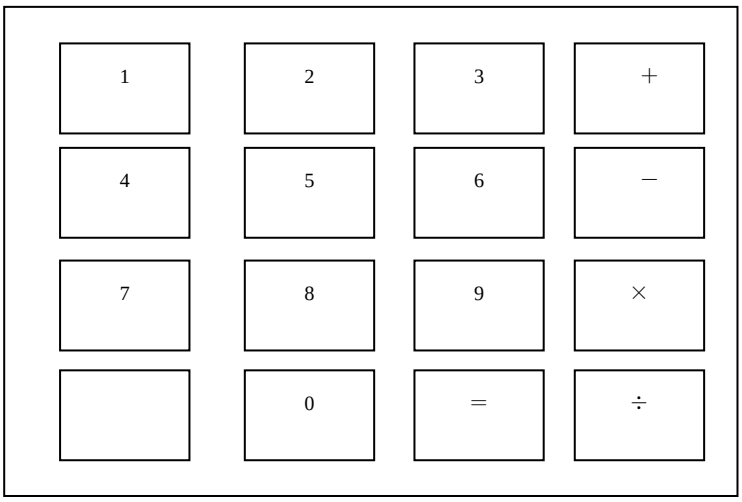


图 4-12 计算器键盘布局

2）运算规则如下：

a）加法：加数和被加数最多 4 位，仅 GLED0 亮表示加法。

b）减法：减数和被减数最多 4 位，不考虑结果为负，仅 GLED1 亮表示减法。

c）乘法：乘数和被乘数最多 4 位，仅 GLED2 亮表示乘法。

d）除法：除数和被除数最多 4 位，结果低四位输出商、高四位输出余数，仅 GLED3 亮表示除法。

e）不考虑连续运算。

f）运算错误，含减法结果为负，最左数码管 A0 显示为‘E’。

3）输入输出规则：

a）输入的操作数为 4 位，多于 4 位则整体左移，丢弃最先输入的，新输入的放在 A0，A3~A0 依次对应千、百、十、个位。

b）当按下操作符后，开始输入第二个操作数，且第一个数为被减数、被除数、被乘数或者被加数。

c）按下“＝”号，才会输出计算结果，此时再按下任意键启动下一轮的计算

3. 实验预习

给出程序的流程图。

4. 实验步骤

参见 4.1.1 的实验步骤。

4.5.2　猜数游戏

1. 实验目的

综合上面实验的训练结果，进一步汇编编程能力。

2. 实验内容

首先通过键盘输入两个数字组成一个十进制两位数（00~99），按下 A 键隐藏该数，然后开始猜数游戏，由键盘输入一个猜测的数，如大于隐藏的数 YLED7 亮，小于则 GLED7 亮，此时可继续输入猜测的数直到猜对为止。猜对后，高 4 位数码管（A7~A4）显示猜对的数，低 4 位数码管（A3~A0）该显示猜对所用的次数。

规则如下：

1）输入状态时，输入的结果数必须小于 3 位，即小于 100，多于 2 位则整体左移，丢弃最先输入的，新输入的放在 A4，A5~A4 依次对应十、个位。

2）输入状态时，A3~A0 为 FFFF

3）输入状态时，输入‘A’后进入猜数状态，此时数码管 A7~A4 为 FFFF 隐藏结果数。

4）猜数状态时，每输入 2 个数字当做一次有效的猜数，并且猜数次数增加一次。

5）猜数状态时，A5~A4 显示猜的数，A3~A0 显示猜的次数。

6）猜数状态时，当所猜数大于隐藏数则 YLED7 亮，小于则 GLED7 亮，猜中则 LED 全灭。

7）猜数状态时，输入‘A’直接在 A5~A4 上显示出结果数，输入 0~A 任意键启动下一次猜数游戏,进入输入状态。

8）猜数状态时，猜中时输入 0~A 任意键启动下一次猜数游戏,进入输入状态。

3. 实验预习

给出程序的流程图。

4. 实验步骤

参见 4.1.1 的实验步骤。

第 5 章 S86 接口实验

本章采用 Minisys-1 实验板，在 S86 系统上进行接口实验，实验中请从硬件和软件两个方面考虑设计与实现的细节。

5.1　S86 系统构建及地址译码器设计

1. 实验目的

熟悉 S86 接口实验系统的构成，学会建立 S86 接口实验初始工程；理解和学会地址译码器设计及简单接口电路的连接，熟悉从 CPU->接口电路->外设的完整过程；学会 S86 汇编程序的编译和可执行代码生成的过程；学会仿真与下载。

2. 实验内容

1) 完成 S86 接口实验初始工程的建立。
2) 完成地址译码器的设计，连通 S86 地址译码器和简单外设（拨码开关和 LED）。
3) 对 S86 汇编程序进行编译和可执行代码生成及下载。

3. 实验要求

按照实验步骤完成实验达到以下要求：

1) 熟练掌握构建 S86 系统完整工程方法。
2) 深入理解地址译码原理和方法。
3) 熟练掌握 Vivado 仿真、综合、实现、生成比特流和下载的全过程。
4) 掌握 S86 系统汇编实验设计、编译与硬件相结合的方法。
5) 掌握 IP 核使用方法。

4. 实验预习

1) 复习本讲义第 2 章和 4.1 节的内容。
2) 理解 8088, 8086 和 S86 的区别。
3) 复习译码器的原理

5. 实验步骤

1） 建立S86 接口实验初始工程

（1） 创建一个项目

双击 

打开 Vivado 2017.4，然后点击 Create Project>，创建一个新项目（或者在菜单栏选择 File->New Project…)。图 5-1 所示。

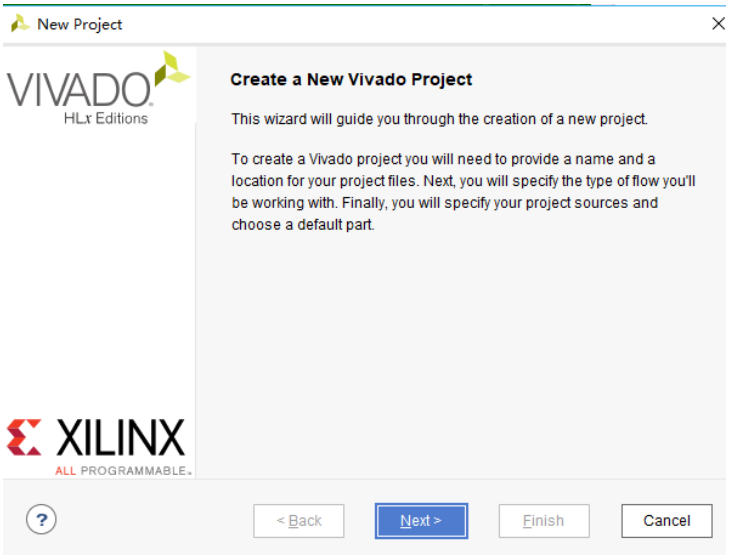


图 5-1 New Project

点击 **Next**。显示如图 5-2 所示的界面。按图 5-2 中所示命名项目名称和路径。这里项目名称为 Ex_1，项目的位置是 C:/sysclassfiles/interface/Ex_1，点击 **Next**。最后，整个项目将在 C:/sysclassfiles/interface /Ex_1/Ex_1 中。

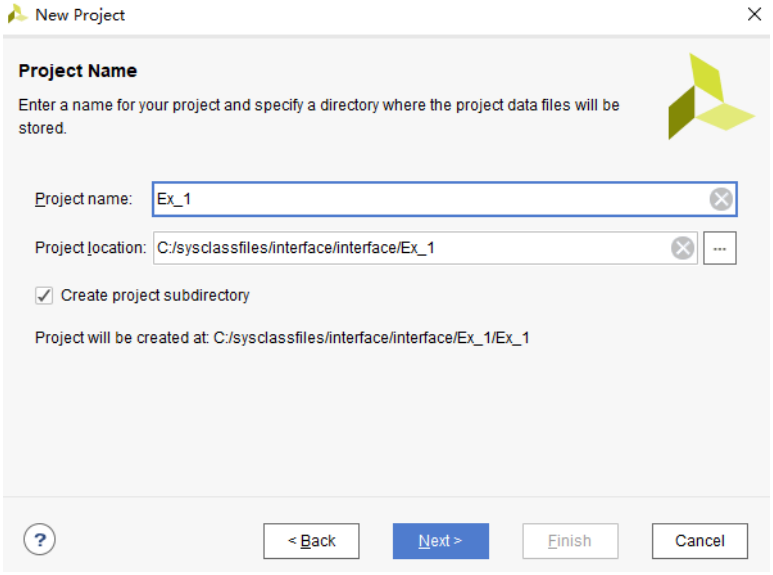


图 5-2 项目名称

如图 5-3 所示设置选择项目类型，红框处点钩，跳过后续步骤，点击 **Next**。

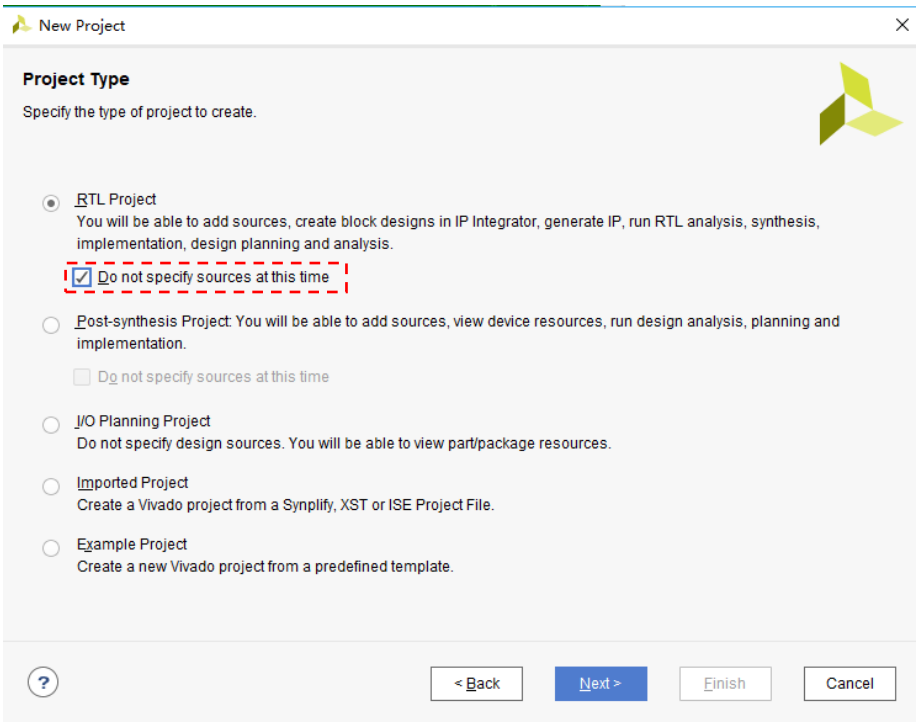


图 5-3 项目类型

按图 5-4 选择器件为 **xc7a100tfgg484-1**。点击 **Next**。

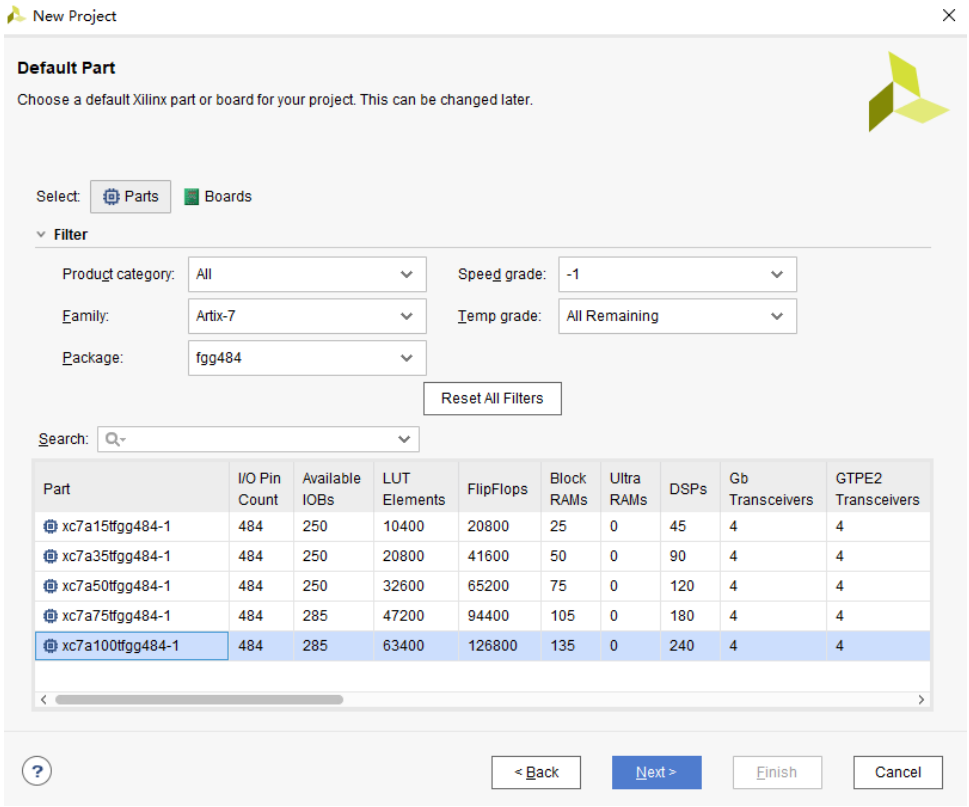


图 5-4 选择器件

可以看到如图 5-5 所示的新项目概览。点击 **Finish**。

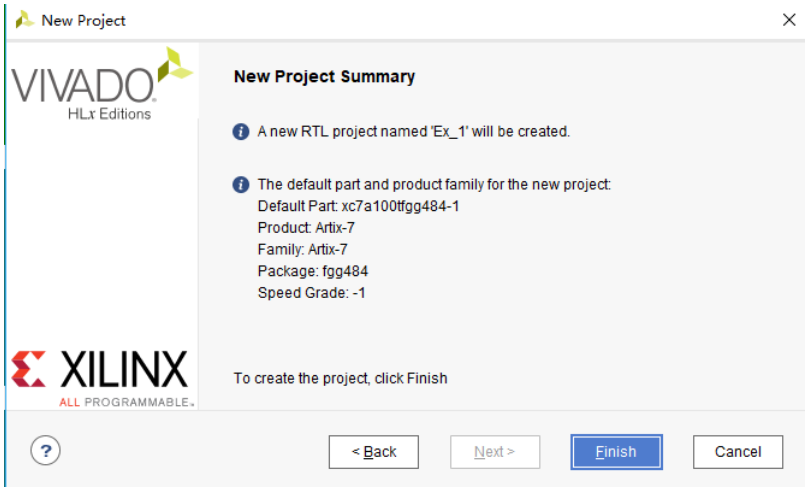


图 5-5 新项目概览

（2） 添加源代码文件

在图 5-6 所示的窗口中右键点击 Design Sources（图中高亮处），在弹出的菜单中选择 Add Sources…，或者使用快捷键 ALT+A，出现图 5-7 所示的对话框。

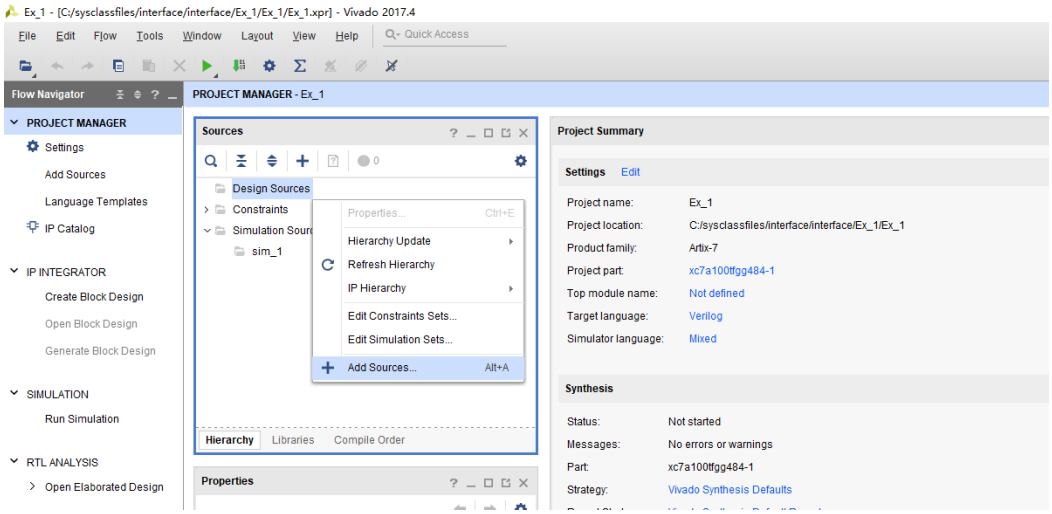


图 5-6 创建新项目后的界面

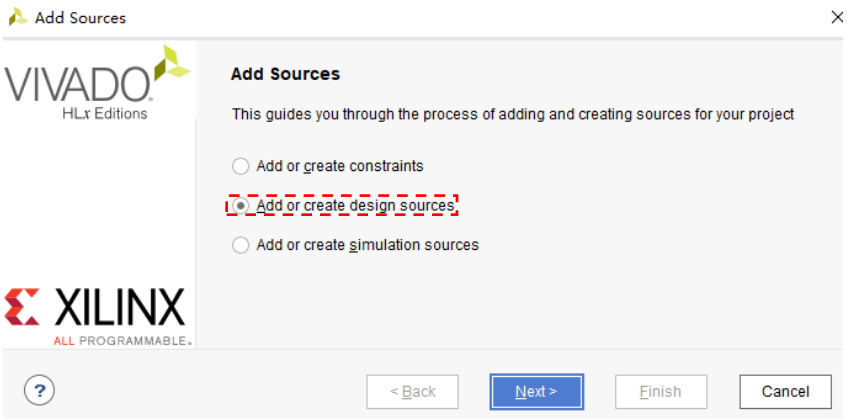


图 5-7 添加源程序对话框

按照图 5-8 所示选择 Add or create design sources 后点击 Next。在接下来打开的对话框

（如图 5-8 所示）中点击 Add Directories。

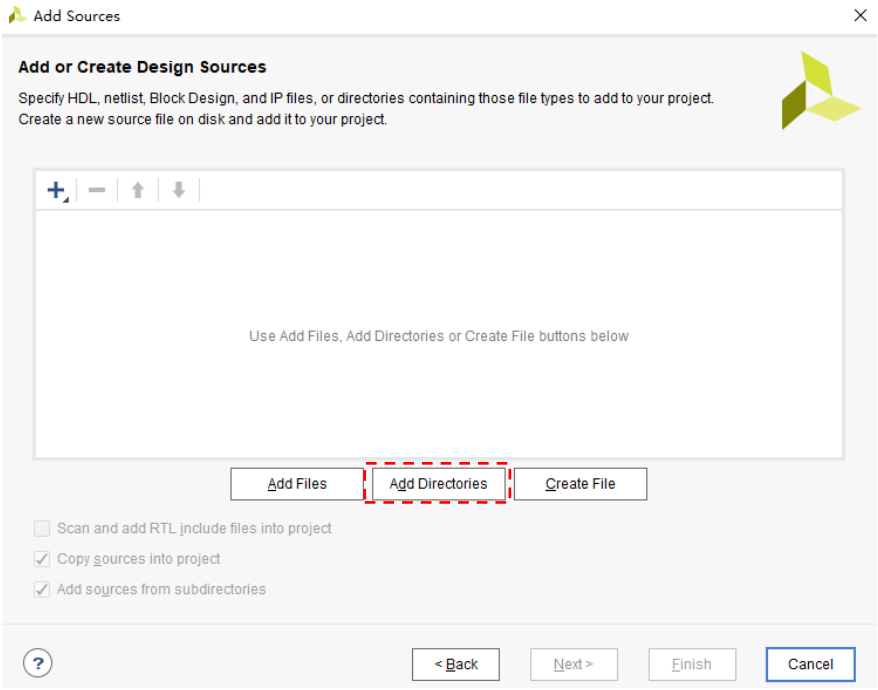


图 5-8 添加或创建设计文件对话框

选择路径为 C:\sysclassfiles\interface\interface\Ex_1\rtl_up，此时会看到所选目录被加入，名字为 rtl_up，位于 C:\sysclassfiles\interface\interface\Ex_1，不勾选 Copy source into project，如图 5-9。点击 Finish。

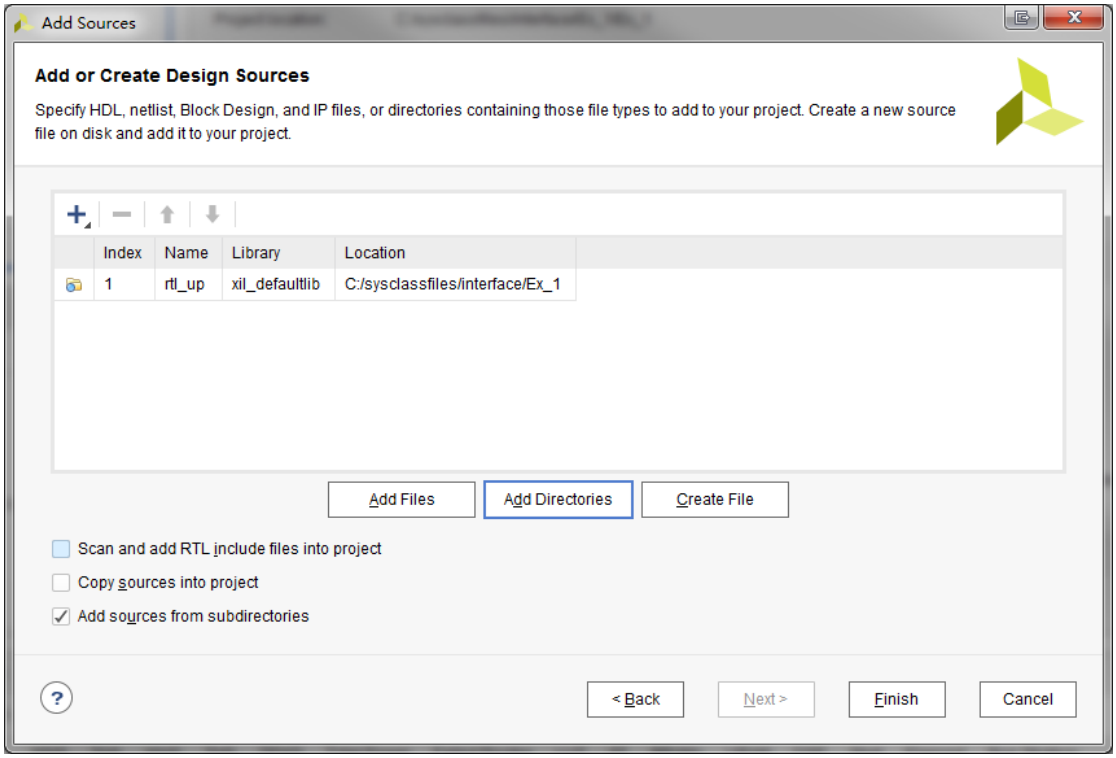


图 5-9 目录加入后添加或创建设计文件对话框

（3）　添加引脚约束文件

图 5-10 所示的窗口是按照上述步骤已经添加入源文件的界面。右键点击 Constraints（图中高亮处），在弹出的菜单中选择 Add or create constraints，用以添加约束文件，如图 5-11 所示。

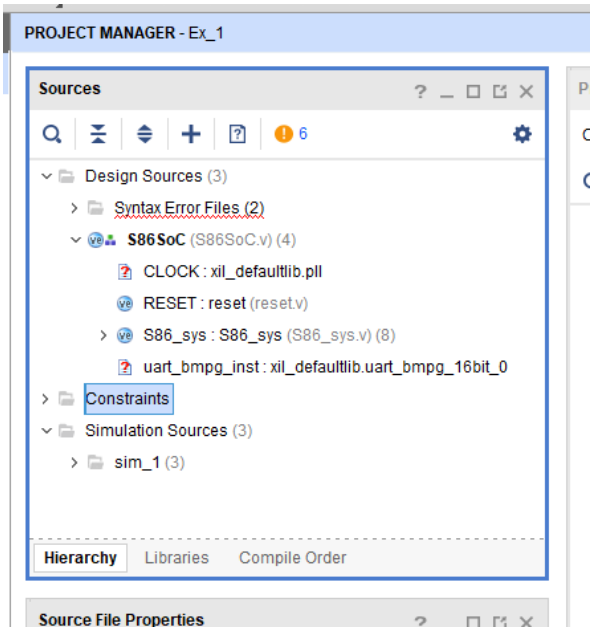


图 5-10 添加源文件后的界面

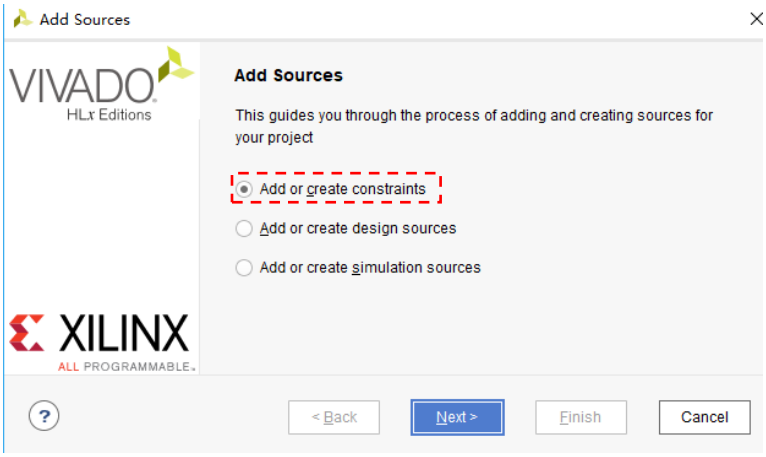


图 5-11 添加引脚约束对话框

点击 Next，选择 Add Files，如图 5-12 所示。

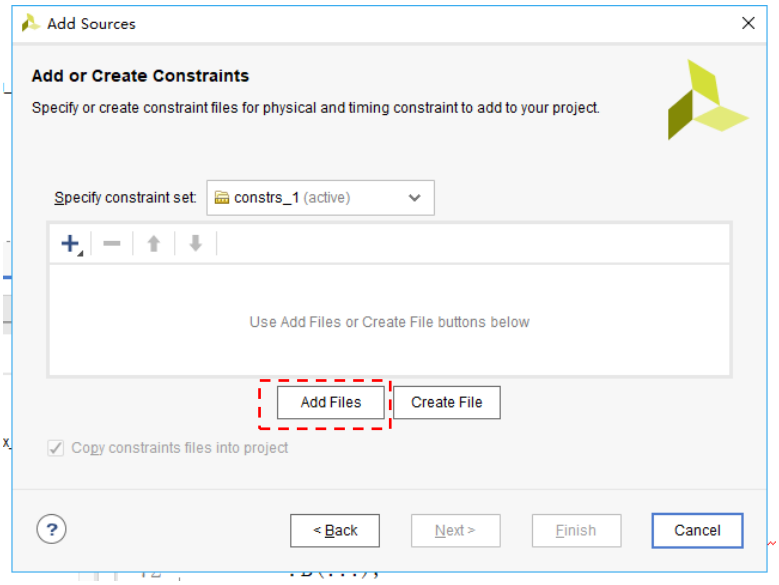


图 5-12 添加或创建引脚约束文件对话

选择 C:\sysclassfiles\interface\interface\Ex_1\rtl_up\xdc\S86.xdc，选择完后可以看到如图 5-13 所示的对话框。点击 Finish。

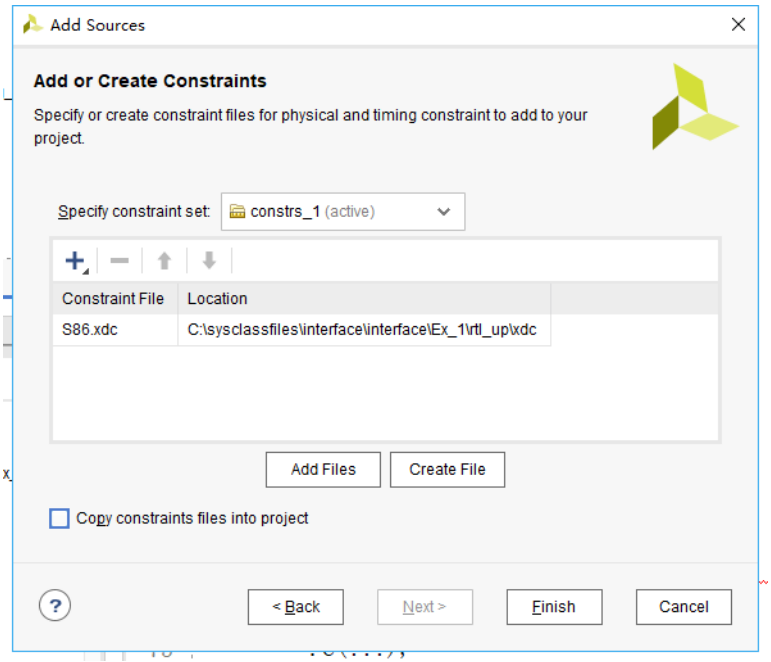


图 5-13 xdc 文件选择后添加或创建设计文件对话框

（4） 导入并添加IP核

源文件和引脚约束文件添加后，还需添加必需的 IP 核。Minisys-S86 系统有 4 个必须的 IP 核，一个 S86 处理器 IP 核、一个是 PLL 锁相环 IP 核用以产生时钟，一个是 Block Ram IP 核用作系统内存，还有一个 UART_bmpg IP 核作为软件程序加载的串行加载部件。其中 S86 处理器 IP 核和 UART_bmpg IP 核需要导入，另外两个均是 Vivado 注册时自带的，只需添加即可。

a) 导入 S86 处理器 IP 核与 UART_bmpg IP 核

点击 Settings（红框所示），如图 5-14 所示。

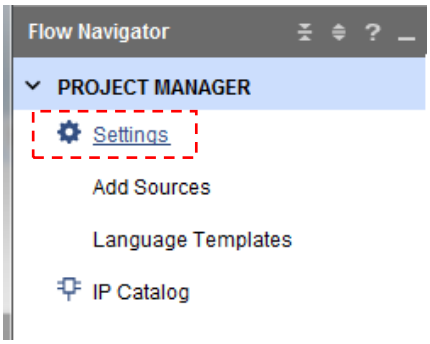


图 5-14 Settings

进入如图 5-15 所示的界面，依次选择 IP，选择 Repository。

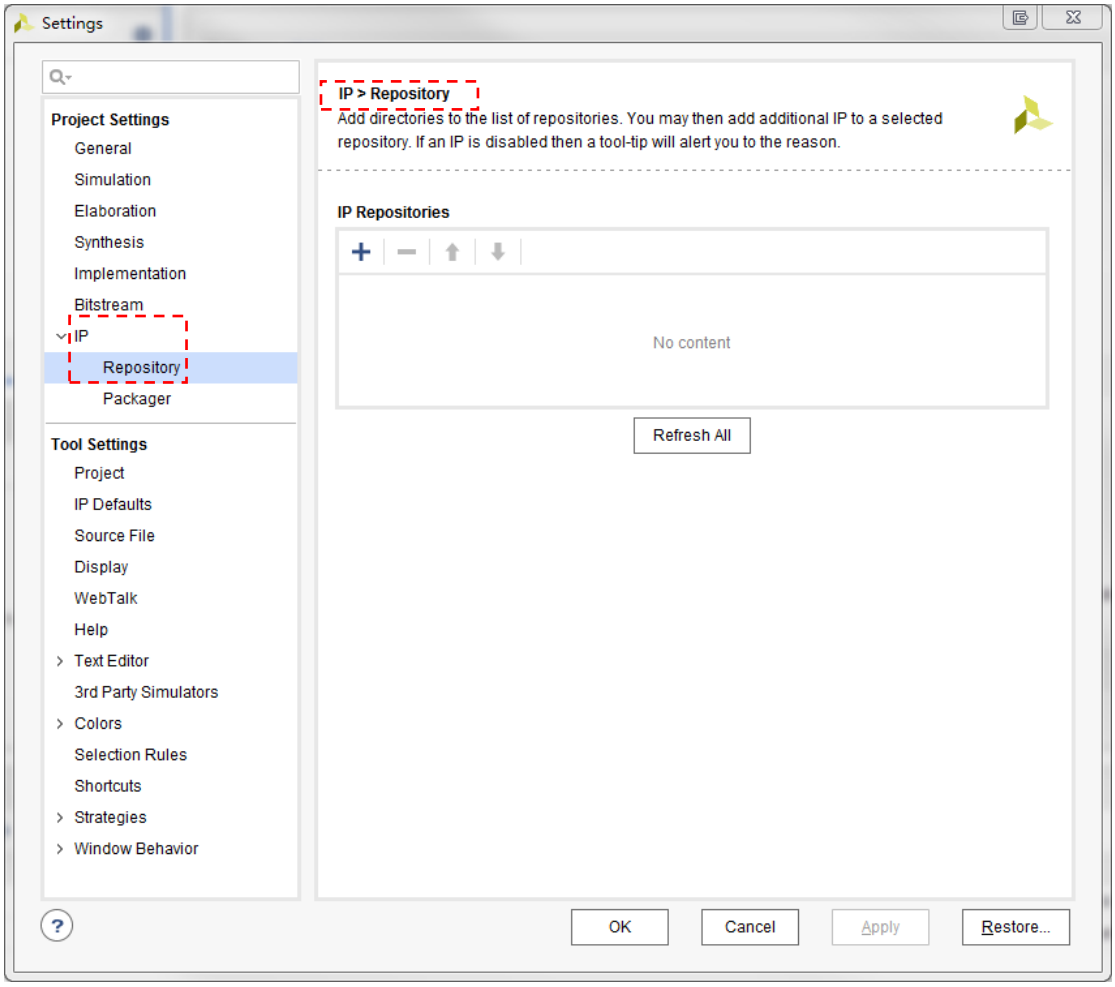


图 5-15 IP 核导入界面

点击 +，选择 C:\sysclassfiles\interface\interface\Ex_1\S86_ip，然后会弹出如图 5-16 所示对话框，可以看到里面包含着一个 IP。

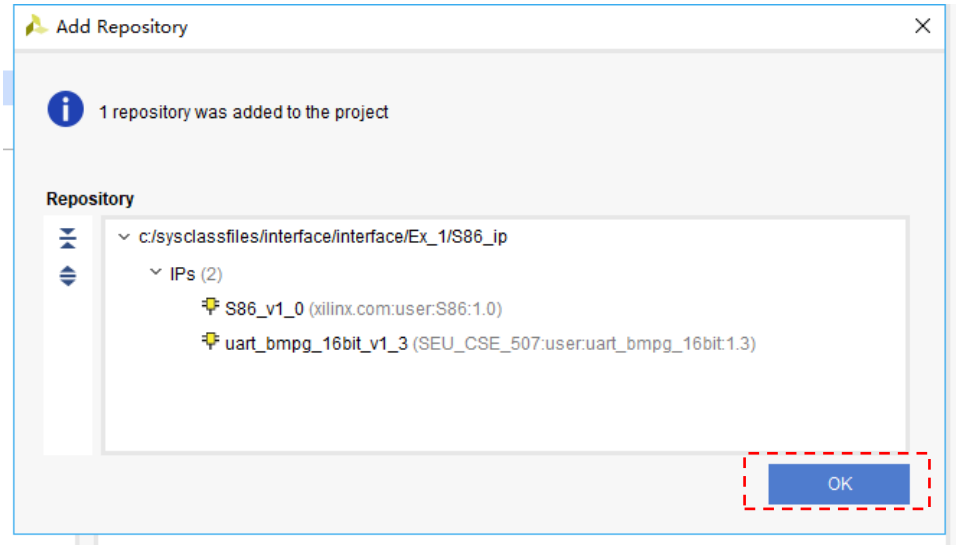


图 5-16 IP 核导入提示对话框

点击 OK，然后，如图 5-17 依次点击 Apply 和 OK。

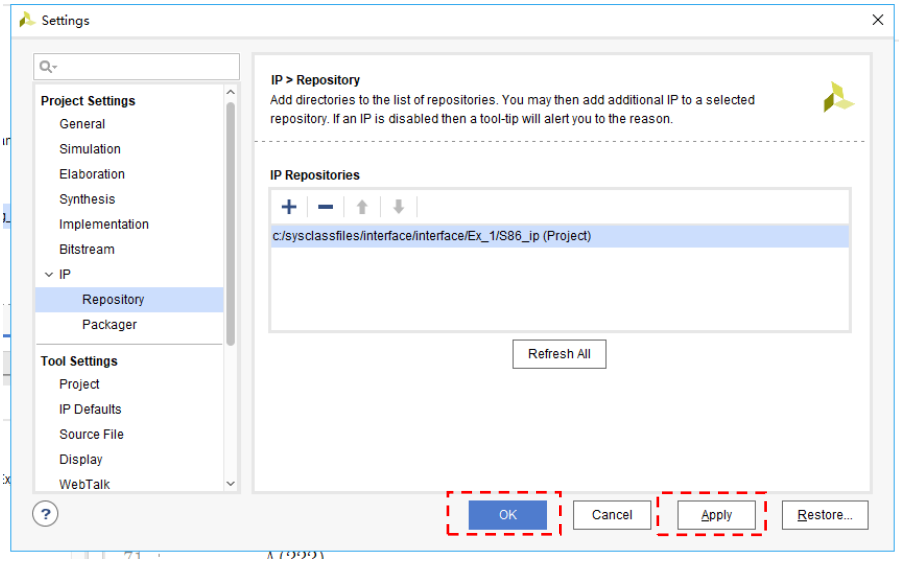


图 5-17 添加 IP 核后对话框

到此，IP 核已经导入，但需要通过添加才能加入到系统中。

b) 添加S86 处理器IP核

首先选择 Project Manager，点击 IP Catalog，如图 5-18 所示。

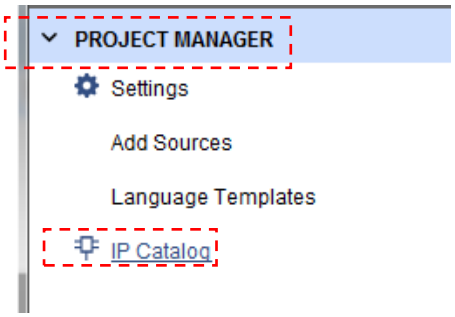


图 5-18 选择 IP Catalog

当出现图 5-19 所示界面时，选择 User Repository ，选择 UserIP。

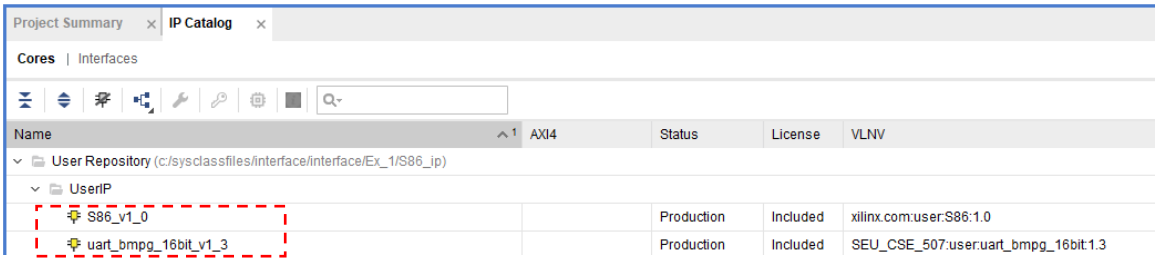


图 5-19 添加 IP 核

双击 S86_v1_0，此时会出现如图 5-20 所示的对话框 ，可以对 S86 IP 核进行设置，这里只需对名字进行修改为 S86_0，点击 OK。

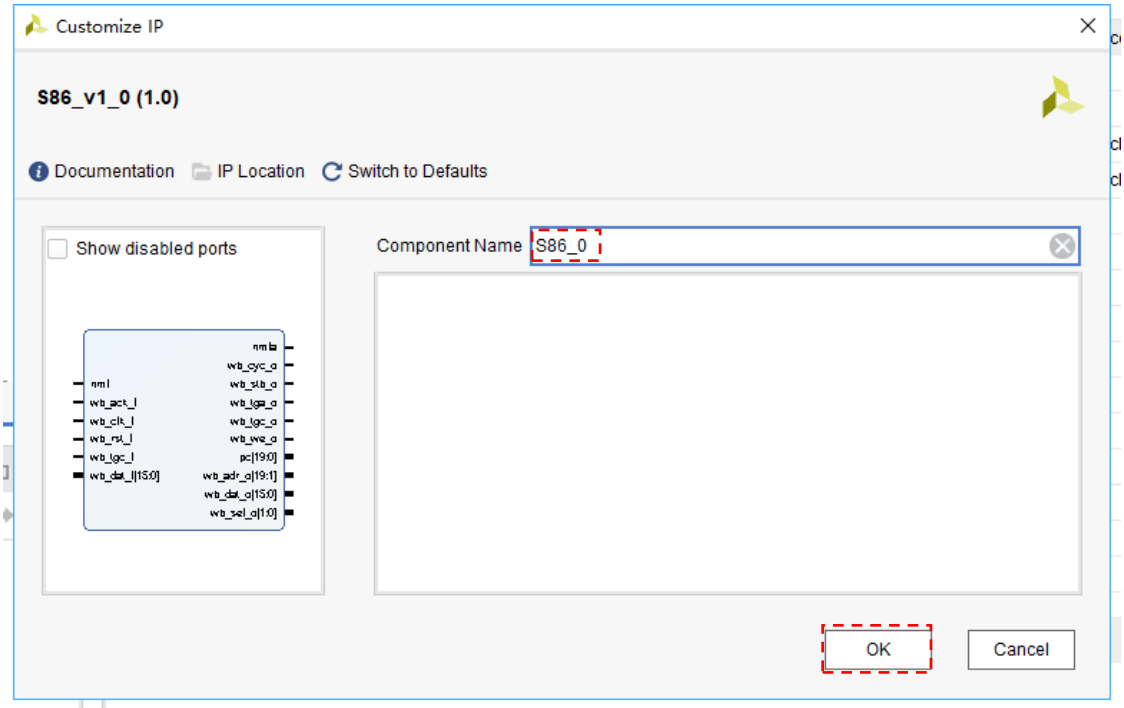


图 5-20 S86 IP 核设置对话框

然后在出现的对话框中点击 OK，接着会出现如图 5-21 所示的对话框，注意勾选 Out of context per IP，点击 Generate。

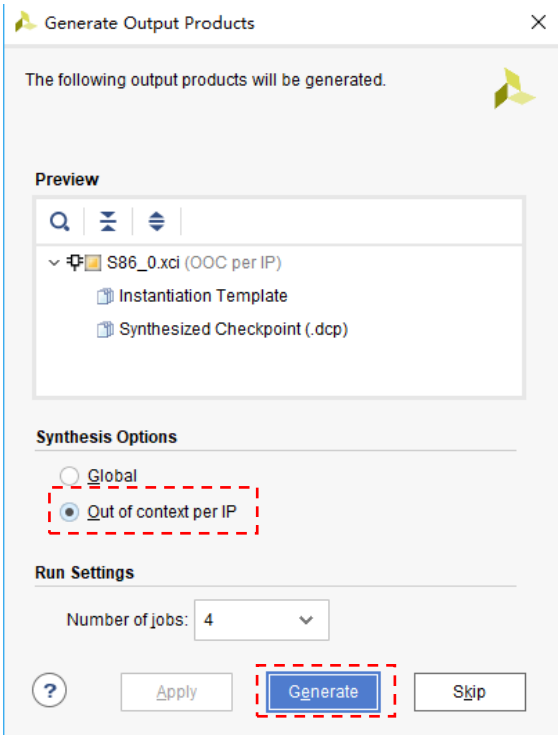


图 5-21 生成 IP 对话框

上述步骤一切正常，将会得到如图 5-22 所示的对话框，点击 OK，到此 S86 处理器加入了系统中。

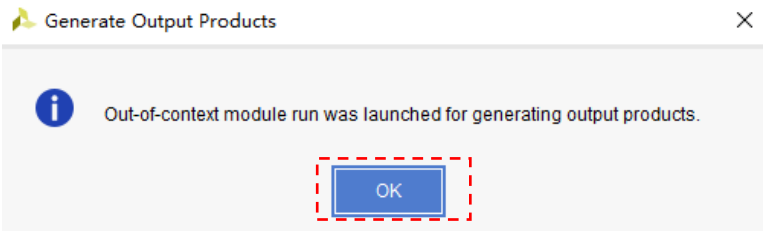


图 5-22 提示对话框

c) 添加UART_bmpg IP核

首先选择 Project Manager，点击 IP Catalog，当出现图 5-19 所示界面时，选择 User Repository ，选择 UserIP。

双击 uart_bmpg_16bit_v1_3，此时会出现如图 5-23 所示的对话框 ，按照图 5-23 对 uart_bmpg IP 核进行设置，点击 OK。

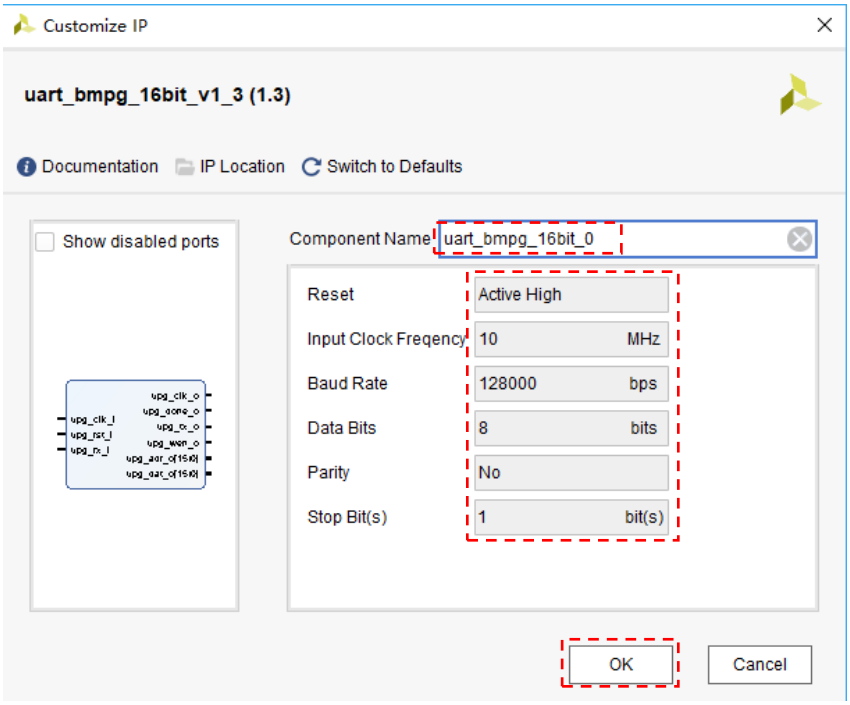


图 5-23 uart_bnpg IP 核设置对话框

然后在出现的对话框中点击 OK，接着会出现如图 5-24 所示的对话框，注意勾选 Out of context per IP，点击 Generate。

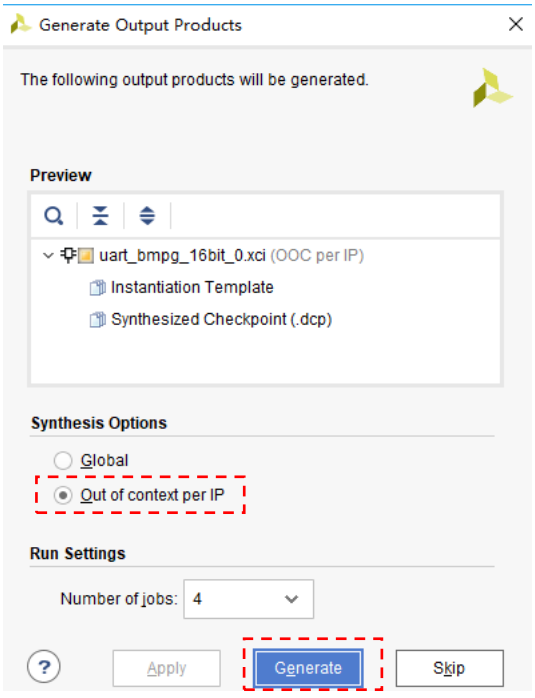


图 5-24 生成 uart_bmpg IP 对话框

上述步骤一切正常，将会得到如图 5-22 所示的对话框，点击 OK，到此 uart_bmpg 处理器加入了系统中。

d) 添加PLL　IP核心

由于 Minisys-S86 系统使用的 PLL IP 核为 Vivado 注册后自带的，所以不需要导入过程，

直接添加就行。

首先，类似于 S86 处理器 IP 核的添加，选择 Project Manager，点击 IP Catalog，如图 5-25 所示。

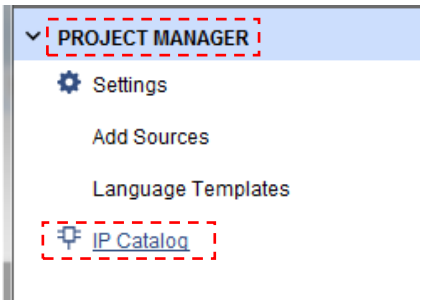


图 5-25 选择 IP Catalog

然后，在出现的 IP Catalog 界面中的 Search 中输入 Clock，选择位于 FPGA Features and Design /Clocking 分支下的 Clocking Wizard，并双击，如图 5-26 所示。

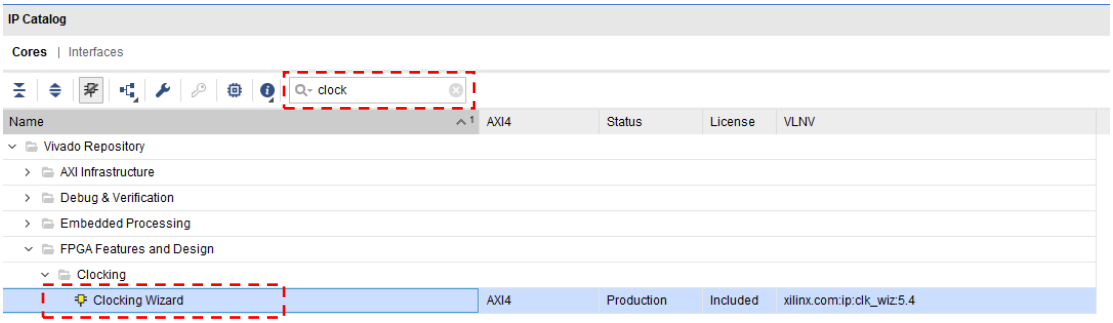


图 5-26 选择 IP 核

类似于 S86 处理器的界面，会出现该 IP 核参数的选择，我们先将名字修改为 PLL，对于一页配置--Cloking Options，在 Primitive 下选择 PLL，其它的参数采用默认的，如有不同按照图 5-27 所示修改。

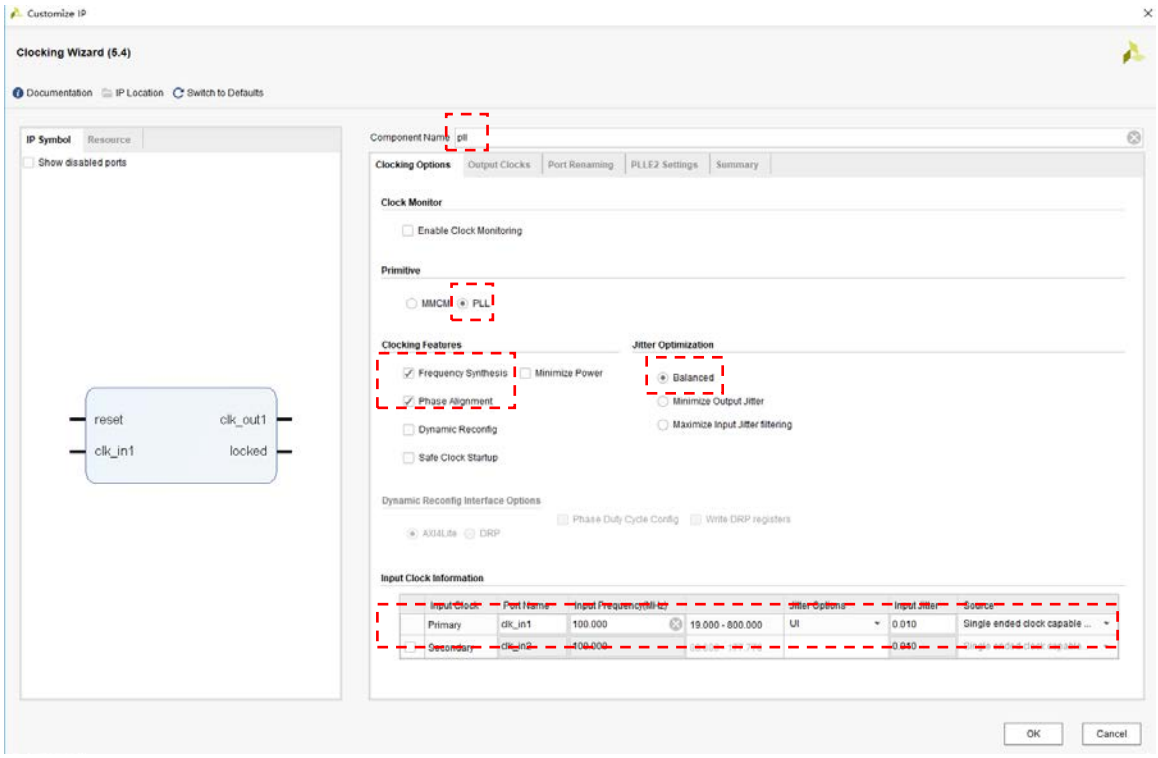


图 5-27 Cloking Options 配置

接着点击 Output Clocks 进行配置，将 clk_out1 的输出频率改成 10MHZ（CPU 时钟和串口时钟），勾选 clk_out2 输出频率为 20MHZ（内存时钟），在 Enble Optional Inputs/Outputs 下取消掉 reset，勾选 locked，如图 5-28 所示

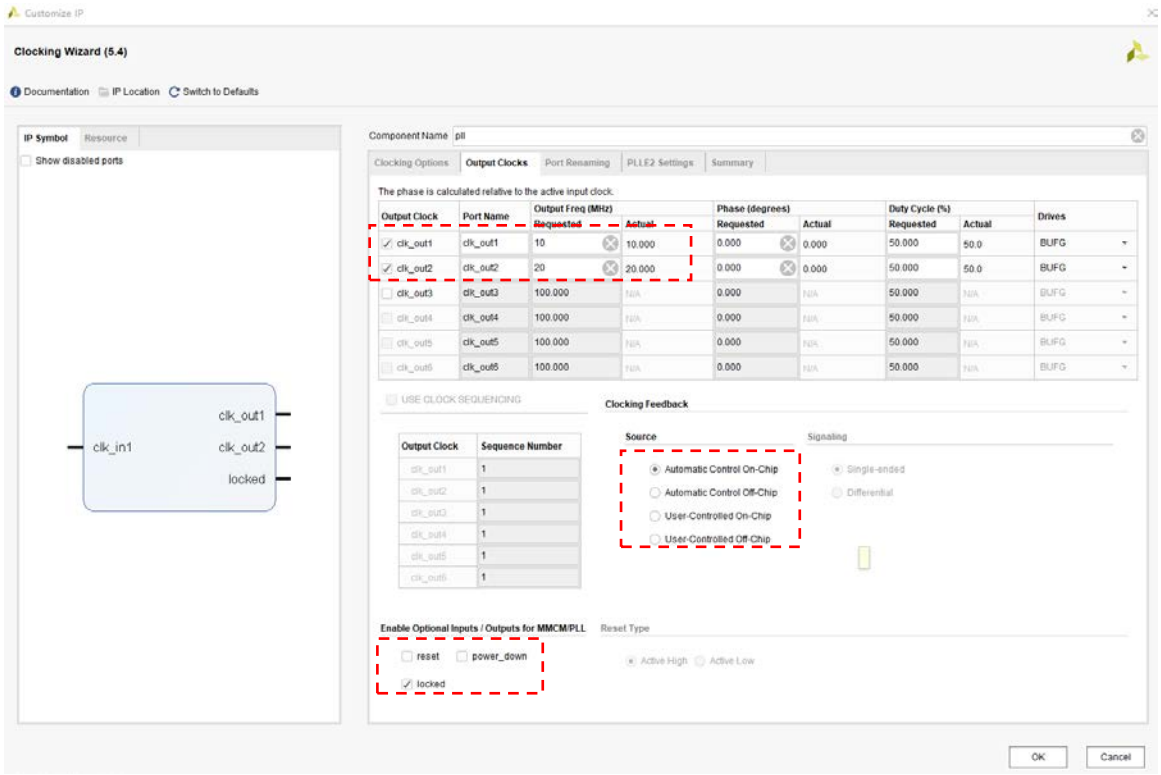


图 5-28 Output Clocks 配置

其它的页面使用默认值，直接点击 OK，然后依照图 5-21 所示生成 IP 核。同理出现如

图 5-22 所示的提示框时，PLL IP 核已经加入了系统中。

e) 添加Block Ram IP核

如图 5-25 所示，选择 Project Manager，点击 IP Catalog。

然后，在 IP Catalog 界面中的 Search 处输入 BRAM 选择 RAM&ROMS&BRAM 分支下的 Block Memory Generator，并双击。如图 5-29 所示。

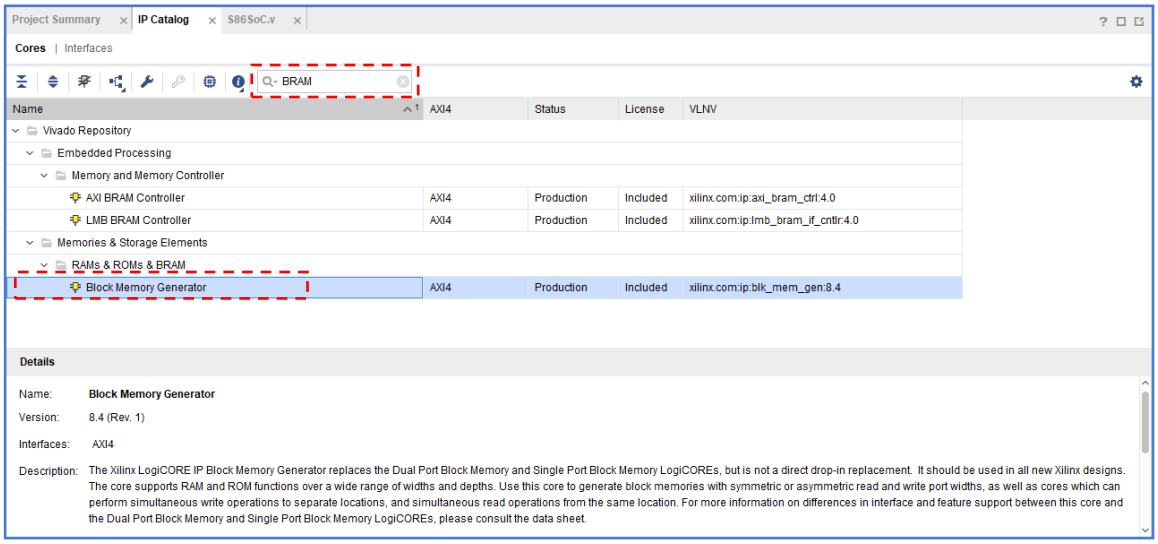


图 5-29 选择 Block Memory Generator

同理，先对一页--Basic 进行配置，先将名字修改为 bios_ram。然后，对于 Interface Type 选择 Native，对于 Memory Type 选择 Single Port RAM，并且勾选 Byte Write Enable，选择 Byte Size(bits)为 8。如图 5-30 所示。

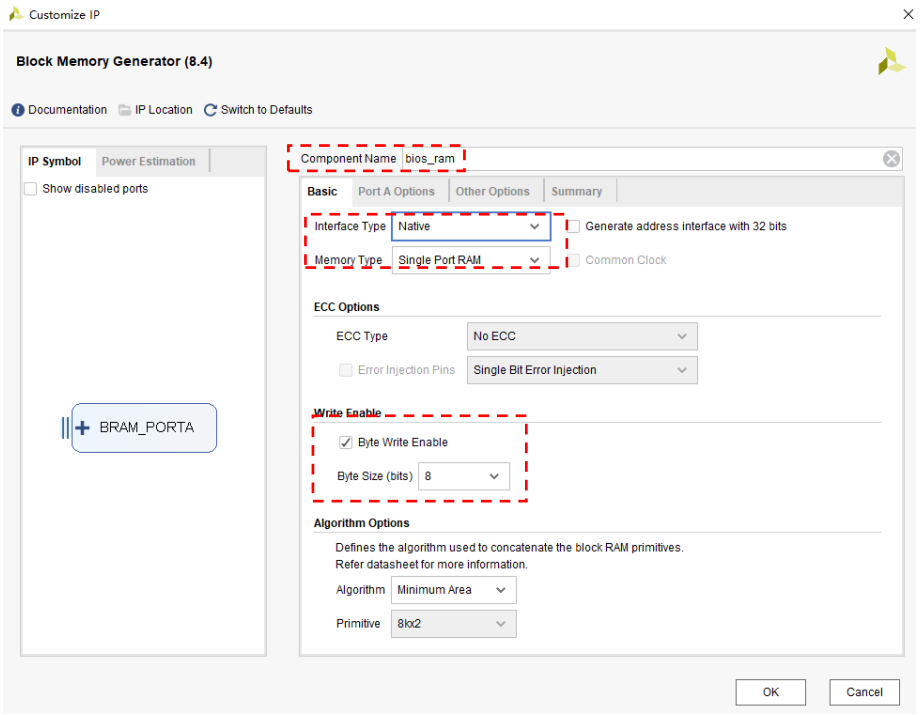


图 5-30 Basic 配置

接着对 Port A Options 进行配置，选择 Write Width 为 16（Read Width 默认相同），Write

Depth 为 65536（Read Depth 默认相同）。由此可知内存大小为 128KB。然后将 Primitives Output Register 选项取消。如图 5-31 所示。

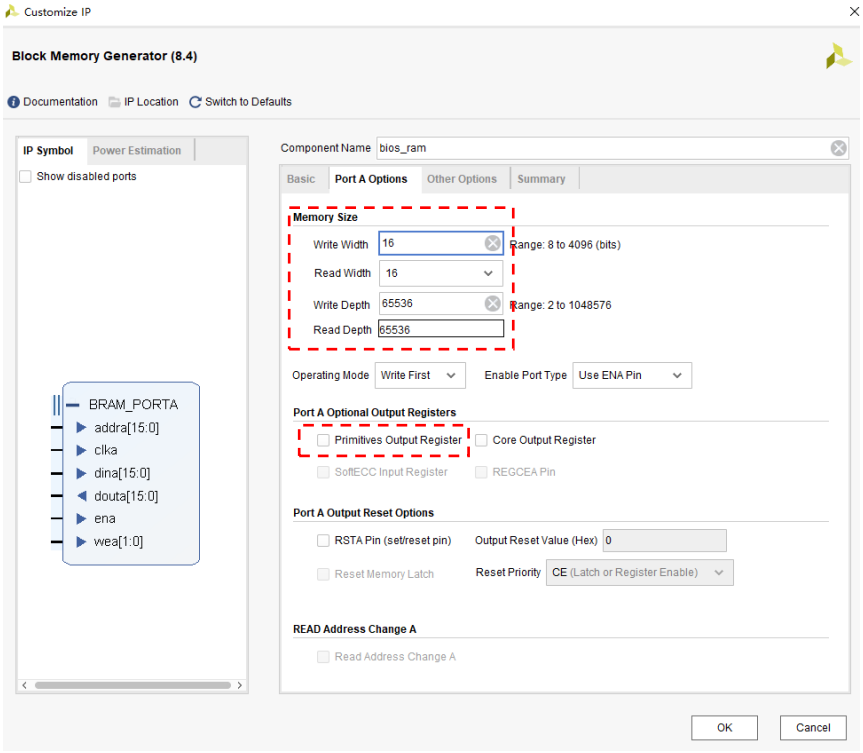


图 5-31 Port A Options 配置

然后，如图 5-32 所示，对 Other Options 页进行配置，勾选 Load Init File，然后通过 Browse 选择 C:\sysclassfiles\interface\interface\Ex_1\rtl_up\initfile\init.coe，**这里的 init.coe 就是内存初始化文件，也就是编写的汇编经过转化最终加载入内存的形式**。

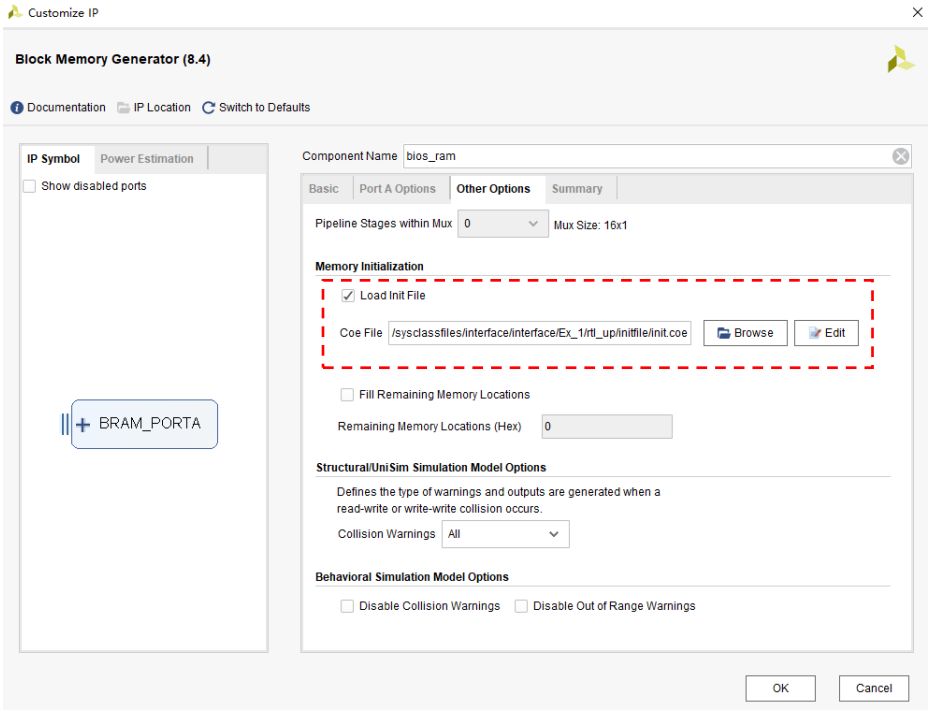


图 5-32 内存初始化文件的加载

最后点击 OK，依照图 5-21 所示生成 IP 核。同理出现如图 5-22 所示的提示框时，**Block Ram** 核已经加入了系统中。

（5）　总结

经过上述步骤，我们已经知道了如何建立一个 S86 接口实验的初始工程。首先，我们需要创建一个项目，然后添加对应的源代码，添加引脚约束文件，最后，导入并添加所需的 IP 核。

2） 设计地址译码器，连通S86 地址译码器和简单外设（拨码开关和LED）

开始设计之前，先回顾下 2.4 中，地址译码器的相关介绍。

对于 2.4.1，介绍了 I/O 地址的安排，同时说明了 Minsys-S86 系统只需使用 10 位地址信号，**采用地址线的第 9 位到 4 位作为地址译码的高端地址**，第 3 位到第 0 位作为低端地址。

对于 2.4.2，地址译码器的连接示意图如图 2-5 所示，

地址译码器通过对从读写信号发生器获取的I/O 读写信号和从S86 处理器获取的高位地址进行译码产生 PIC 片选信号和外设片选信号，而应答信号发生器需要判断片选信号产生应答信号，输入数据多路选择器中涉及到对片选信号的判断来进行多路选择。

（1）　设计地址译码器

由之前的描述：地址译码器通过对从读写信号发生器获取的 I/O 读写信号和从 S86 处理器获取的高位地址进行译码产生 PIC 片选信号给和外设片选信号。我们可以知道地址译码器模块必然有以下几种端口：1、地址信号（输入）；2、I/O 读写信号（输入）；3、I/O 片选信号（输出）。

对于地址信号，Minsys-S86 系统只需使用 10 位地址信号，**采用地址线的第 9 位到 4 位作为地址译码的高端地址**，故应该有 6 位地址信号输入，其中根据外设译码个数来决定哪几位作为具体译码地址。

对于 I/O 读写信号，由读写发生器产生。

对于 I/O 片选信号，此处只涉及到 2 个外设，所以只需产生 2 个输出即可。

由上述分析可知，我们可采用常用的 **3-8 译码器**当做地址译码器。3-8 译码器真值表如表 5-1 所示。

表 5-1 3-8 译码器真值表

输　入				输　出							
G1	G2An	G2Bn	C B A	Y7N	Y7N	Y7N	Y7N	Y7N	Y7N	Y7N	Y7N
1	0	0	0 0 0	1	1	1	1	1	1	1	0
1	0	0	0 0 1	1	1	1	1	1	1	0	1
1	0	0	0 1 0	1	1	1	1	1	0	1	1
1	0	0	0 1 1	1	1	1	1	0	1	1	1
1	0	0	1 0 0	1	1	1	0	1	1	1	1
1	0	0	1 0 1	1	1	0	1	1	1	1	1
1	0	0	1 1 0	1	0	1	1	1	1	1	1
1	0	0	1 1 1	0	1	1	1	1	1	1	1
0	X	X	X X X	1	1	1	1	1	1	1	1
X	1	X	X X X	1	1	1	1	1	1	1	1
X	X	1	X X X	1	1	1	1	1	1	1	1

首先找到地址译码器对应的设计文件，双击 Design Sources 中的 U1-Decoder，打开

Decode.v。如图 5-33 所示。

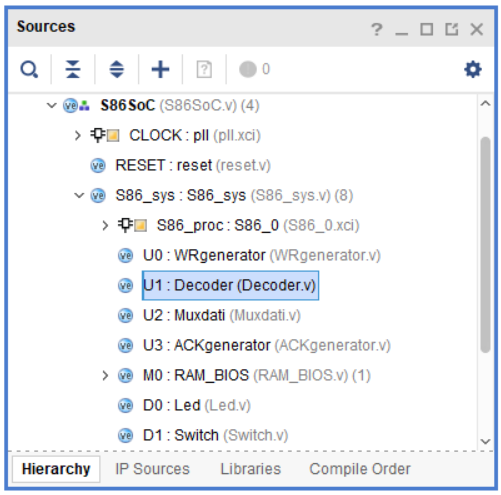


图 5-33 打开 Decoder.v

然后请根据 3-8 译码器真值表完成以下代码，如果数电实验已经做个 38 译码器，可以直接使用。

```
module Decoder (
        input A,
        input B,
        input C,
        input G1,
        input G2AN,
        input G2BN,
        output Y0N,
        output Y1N,
        output Y2N,
        output Y3N,
        output Y4N,
        output Y5N,
        output Y6N,
        output Y7N
        );
       integer i;
      reg [7:0] YN;
       wire[2:0] cba,G;
       assign cba = {C,B,A};
       assign G = {G1,G2AN,G2BN};
     assign Y0N = YN[0];
       ……//定义寄存器用作 always 块输出暂存，并将输出与之相对应
       always @(*)
       begin
       ……//填写 38 译码器功能代码
```

```
        end
endmodule
```

（2）　完成地址译码器端口连接

经过上述步骤，我们设计了一个 3-8 译码器，但是要将它加入 Minisys-S86 系统中，需要完成其端口的连接。

首先找到工程中对应地址译码器端口连接的部分，双击 Design Sources 中的 S86_sys，打开设计文件 S86_sys.v 找到注释有地址译码器的位置，其中的 Decoder 是我们设计的译码器模块，U1 是它的例化名称。如图 5-34 所示。

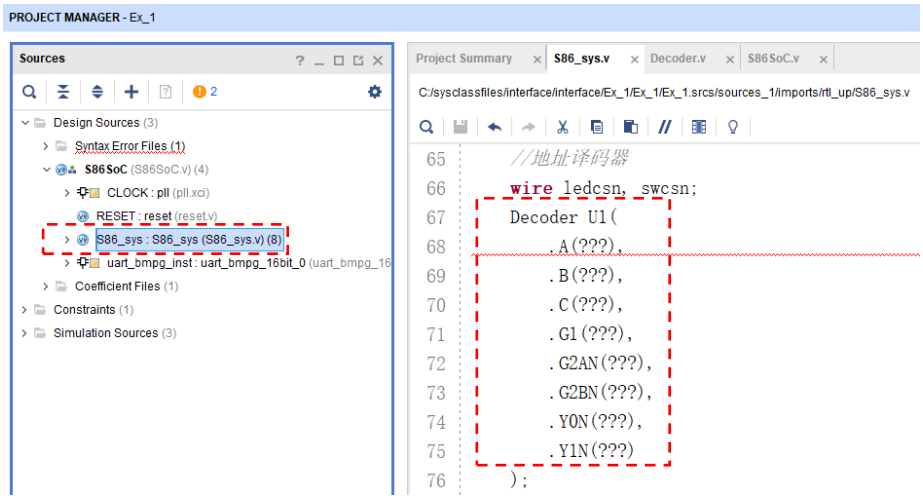


图 5-34 地址译码器端口

端口命名和 3-8 译码器保持一致，但是只有 2 个输出 Y0N 和 Y1N，是因为我们只用到了 LED 和拨码开关（SW）。

接下来我们对整个 S86 系统的接口部件连接做一个比较详尽的介绍，请读者根据以下的介绍所给的提示，在 S86_sys.v 中完成译码器端口的连接，即替换图 5-34 括号中的问号。

a) 连接I/O片选信号

此处需连接的 I/O 片选信号，即 Y0N 和 Y1N。我们将 Y0N 当做 LED 的片选信号，Y1N 当做 SW 的片选信号。

提示：首先定义 wire ledcsn, swcsn;，然后完成连接。

b) 连接地址信号

根据 3-8 译码器的译码，将地址信号填到对应的位置上。地址信号由 S86 处理器产生，对应的地址信号的名称为 adr，在工程中的位置如图 5-35 所示。

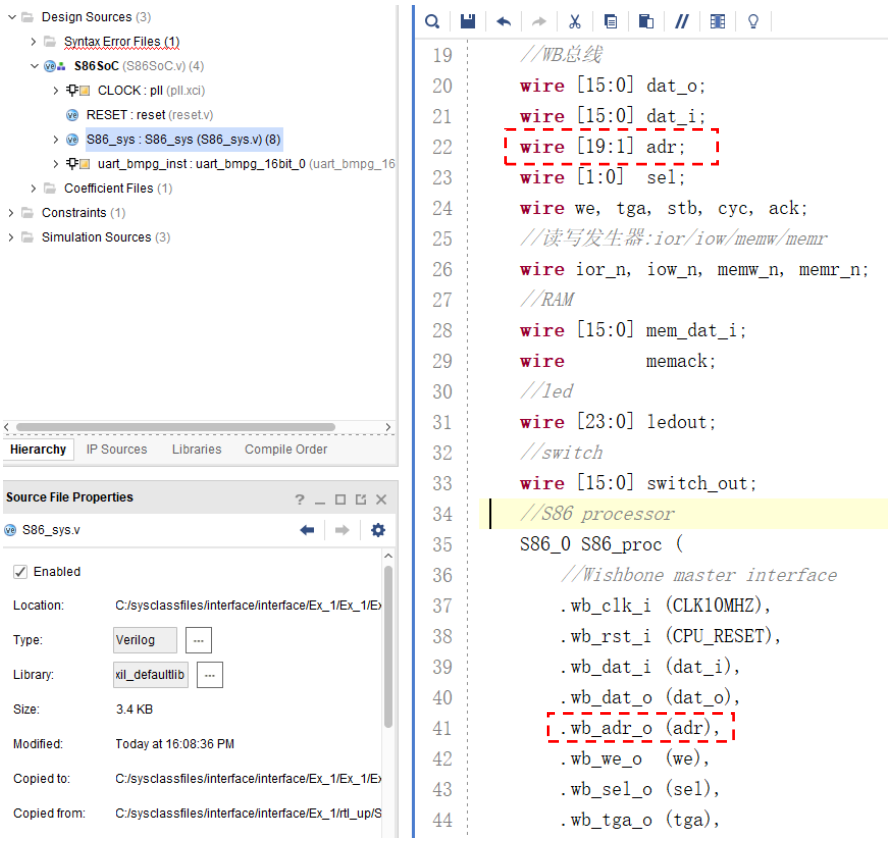


图 5-35 地址信号名称

根据前面的分析我们所需要的地址信号为 adr[4]、adr[5]、adr[6]、adr[7]、adr[8]、adr[9]。将其适当的组合填到图 5-34 相应的括号的???中，使得 3-8 译码器的输出端地址范围符合表 2-5 中的规定，比如 LED 的地址范围是 000000H~00000FH，Switch 的地址范围是 000010H~00001FH。

c) 连接I/O读写信号

I/O 读写信号由读写信号发生器产生，对应的地址信号的名称为 ior_n 和 iow_n，在工程中的位置如图 5-36 所示，其中工程文件中已经定义好了相关的信号如下。

```
//读写发生器:ior/iow/memw/memr
wire ior_n, iow_n, memw_n, memr_n;
```

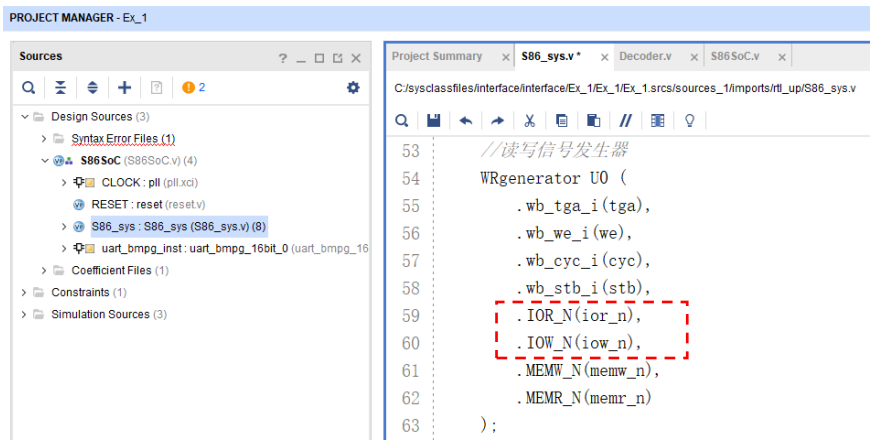


图 5-36 I/O 读写信号

请将 ior_n 和 iorw_n 以合适的方式填到图 5-34 相应的括号中的？？？上。

（3）　LED模块连接

下面将介绍 LED 模块的连接，其他写设备可通过类似方法设计端口和连接。LED 模块端口连接如图 5-37 所示，其中工程文件中已经定义好了相关的信号如下。

```
//led
wire [23:0] ledout;
```

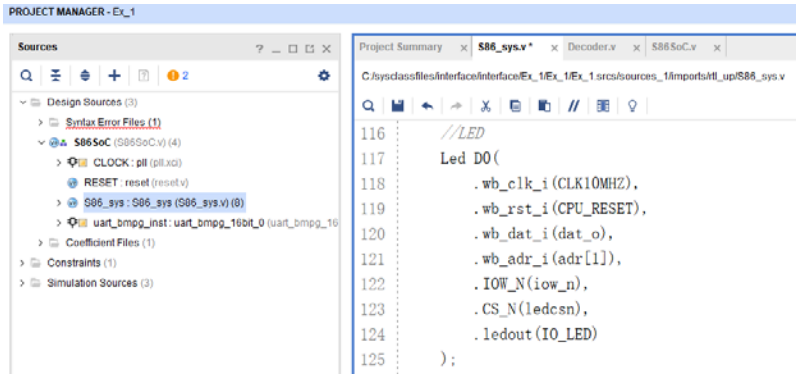


图 5-37 LED 模块端口连接

a) 时钟信号（wb_clk_i）

钟信号主要由两种作用，一种用以驱动外设以保持和 S86 同步，对于异步通信外设则可不需该信号（如 8254）；另一种是由于外设本身有时序要求（如多位数码管的显示）。

此处只是为了与 S86 产生同步。故将系统时钟 clk 接入即可。

b) 复位信号（wb_rst_i）

用以复位，将 rst 接入。

c) 数据信号（wb_dat_i）

对于需要从 S86 中获取数据的外设需要将数据信号接入，该信号来源于 S86 的总线的数据输出信号 dat_o，dat_o 的位宽为 16 位，对于有些外设只需要低 8 位，则只需将 dat_o 的低 8 位连接。

d) 低位地址信号（wb_adr_i）

该信号用以与片选信号配合，用来区分访问的端口。由于只使用偶数地址，而对于 24 位的 LED，使用 16 位宽的数据总线，需要 2 个地址用以区分，而对应表 2-5 的 LED，故此处只需要将 adr[1]连接即可。

此处简化了地址的计算方式，因为本讲义所设计的 I/O 地址为 10 位，其余位默认为 0，而不同外设的 I/O 地址对应片选信号的是地址的高 6 位，对于原本的第 3 位到第 0 位作为低端地址简化为 1 位即可区分出。

译码出的地址构成如表 5-2 所示。

表 5-2 译码地址构成

高 10 位	片选信号对应 6 位地址	低位地址 3 位	1 位
0000000000	000000（对应 LED）	00（简化）+ wb_adr_i	0（偶数地址低位为 0）

其中低位地址可以根据外设所需要区分的地址数进行简化，如 LED 只需要区分 2 个地

址所以可以简化为 1 位。

e) I/O写信号（IOW_N）

该信号由读写信号发生器产生，表示进行 I/O 写操作，直接连接 iow_n 即可。对应的如果是进行 I/O 读则需要设计 IOR_N 端口，连接 ior_n 信号。

f) 片选信号（CS_N）

该信号由前面设计的地址译码器产生，低电平表示对当前外设进行访问，将 ledcsn 接入即可。

g) 输出信号（ledout）

即输出到 led 引脚的信号，此处直接连接到板上的 led 灯，即 led_。

（4） 拨码开关模块连接

下面将介绍拨码开关模块的连接，其他读设备可通过类似方法设计端口和连接。拨码开关模块端口连接如图 5-38 所示，其中工程文件中已经定义好了相关的信号如下。

```
//switch
wire [15:0] switch_out;
```

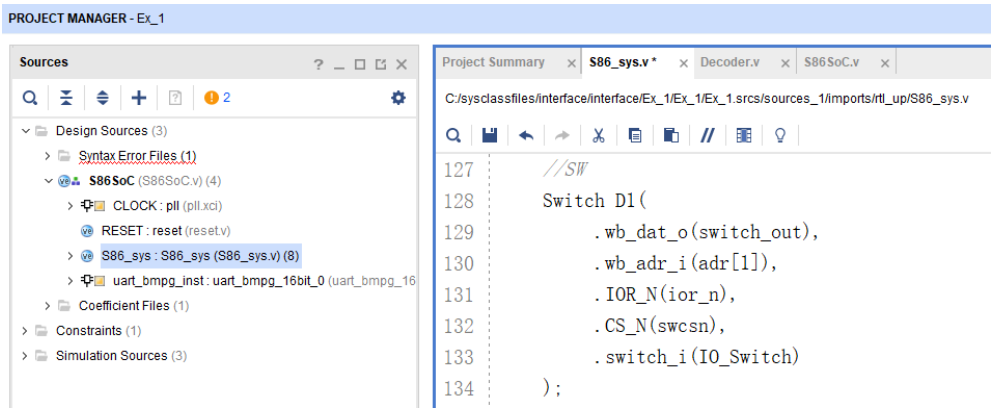


图 5-38 拨码开关端口连接

a) 数据信号（wb_dat_o）

该信号为输出至输入数据多路选择器的数据，由于 S86 数据总线位宽为 16，所以一次总线读周期最多能读取 16 位数据，故先定义 wire [15:0] switch_out，然后连接即可。

b) 低位地址信号（wb_adr_i）

该信号用以与片选信号配合，用来区分访问的地址。由于只使用偶数地址，而对于 24 位的拨码开关，使用 16 位宽的数据总线，需要 2 个地址用以区分，同 LED 模块，此处只需要将 adr[1]连接即可。

此处简化了地址的计算方式，因为本手册所设计的 I/O 地址为 10 位，其余位默认为 0，而不同外设的 I/O 地址对应片选信号为地址的高 6 位，对于原本的第 3 位到第 0 位作为低端地址简化为 1 位即可区分出。

c) I/O读信号（IORN）

该信号由读写信号发生器产生，表示进行 I/O 读作，直接连接 ior_n 即可。

d) 片选信号（CS_N）

该信号由前面设计的地址译码器产生，低电平表示对当前外设进行访问，将 swcsn 接入即可。

e) 输入信号（switch_i）

即从拨码开关引脚的输入的数据，此处直接连接到板上的拨码开关，即 switch_。

（5） 应答信号发生器修改

由于引入新的外设，所以需要在应答信号发生器上将片选信号和相应逻辑加上，添加的端口和相应逻辑如图 5-39 所示。只需要将添加的片选信号取反然后相或即可。

ACKgenerator.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   module ACKgenerator(
4       input MEMACK,
5       input IOCS0_N,
6       input IOCS1_N,
7       output ACK
8       );
9
10      assign ACK = MEMACK | (!IOCS0_N) | (!IOCS1_N);
11
12  endmodule
```

图 5-39 应答信号发生器添加端口和逻辑

同时，在 S86_sys.v 中，对应的端口连接如图 5-40 所示。

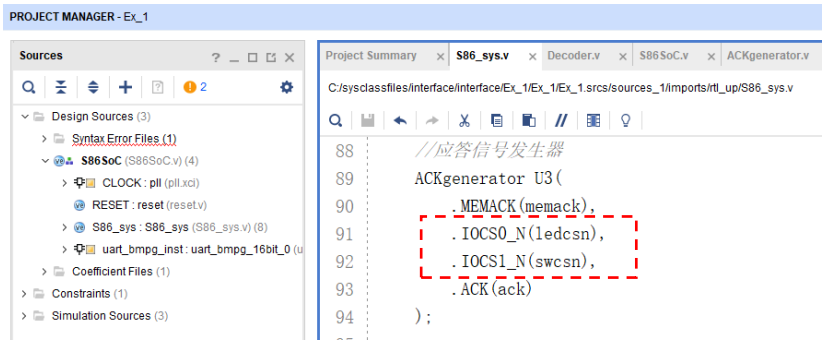


图 5-40 应答发生器端口连接

IOCSx_N 代表着添加外设的片选信号，这里添加了 2 个外设则将 ledcsn 和 swcsn 连接即可。

（6） 输入数据多路选择器修改

由于引进了新的读外设拨码开关模块，所以需要在输入数据多路选择器上加入新的端口和逻辑。添加的端口和逻辑如图 5-41 所示。每引进一个读外设，即需要添加一个该外设的片选信号和相应的外设数据信号，然后补充逻辑进行判断，若片选有效，则让 DATO 等于

对应的数据信号即可。

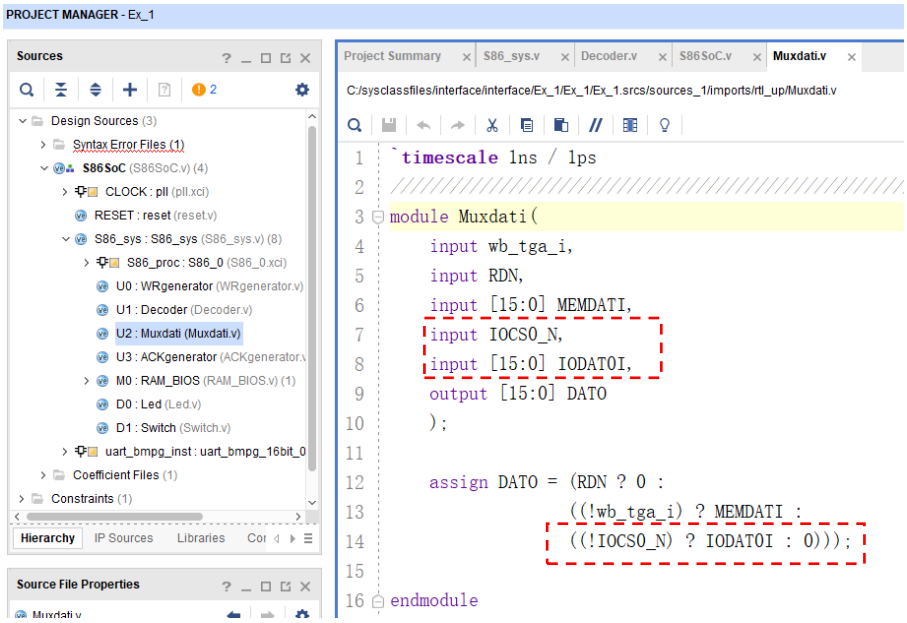


图 5-41 多路输入选择器添加的端口和逻辑

同时，在 S86_sys.v 中，对应的端口连接如图 5-42 所示。

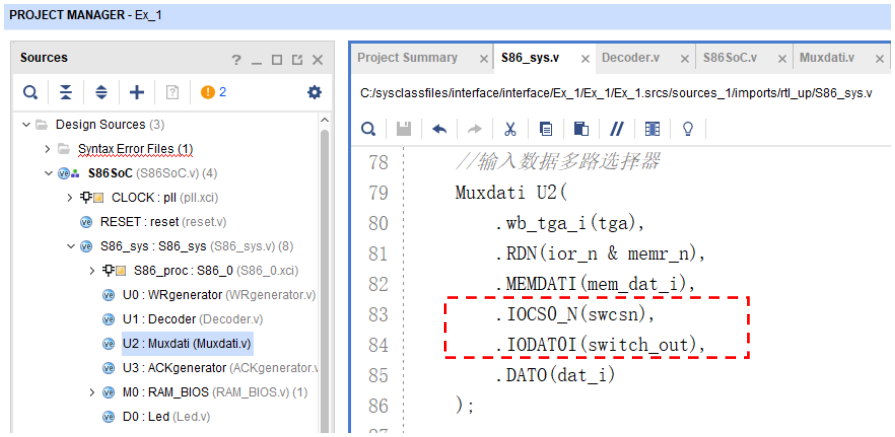


图 5-42 输入数据多路选择器连接

主要连接的为片选信号和对应外设数据信号，在本实验中只有拨码开关，故将拨码开关片选信号（swcsn）和对应数据信号（switch_out）接入即可。

（7）　总结

通过上述步骤，我们知道了如何设计一个地址译码器，同时如何连接简单的外设。需要注意的是当我们每加入一个外设就要在应答信号发生器中加入端口和相应逻辑；每加入一个读外设就要在输入数据多路选择器中加入端口、数据信号和相应的逻辑。

3）　编写S86 汇编，进行编译，转换成coe文件，仿真，生成比特流文件和下载

计算机只能识别二进制代码，因此计算机能执行的指令必须以二进制代码的形式表示，这种以二进制代码形式表示的指令称为指令的机器码（Machine Code）。为便于人们使用，采用汇编语言来编写程序。汇编语言是一种符号语言，它用助记符来表示操作码，用符号或符号地址来表示操作数或操作数地址，它与机器指令是一一对应的。

把源程序翻译成机器语言程序（目标程序）的过程叫做汇编（Assemble）。完成这个过

程的程序叫做汇编程序（Assembler），下面将以微软（Microsoft）公司提供的 MASM 宏汇编程序为样本来设计并编译汇编语言。由于目前 S86 所需要内存映像文件为 coe 形式，所以我们开发了 exe2coe 工具将编译生成的 exe 文件转换成 coe 文件，coe2txt 文件将该 coe 文件转换为能用串口传输的 txt 文件，这样硬件通过综合、实现、下载到板子上之后，可以通过串口将转为 txt 文件的可执行代码加载 CPU 中运行。

为了方便读者使用，我们专门开发了 i8086 IDE 集成开发环境，将上述软件全部集成。

（1）　工具的安装

按照 2.6.1 中工具安装步骤安装工具 i8086 IDE，如果以及安装好，这一步可省。

（2）　汇编程序编写

汇编编写的模板如下所示。

```
DATA SEGMENT 'DATA'
     ;用户数据
DATA ENDS

CODE SEGMENT 'CODE'
   ASSUME CS:CODE, DS: DATA
START:
   MOV AX, 0080H  ;数据段从内存的 00800H 开始，因此 DS=0080H
   MOV DS, AX
   ;用户代码
   JMP START
CODE  ENDS
      END START
```

栈段和数据段都在 00800H 和 0FFFFH 范围内，栈从 0FFFFH 由高地址向低地址增长。S86 数据段起始位置为 00800H 处，所以如果需要数据段，则需要在汇编中设置 DS 为 0080H，而堆栈段寄存器 SS 不需设置。作为汇编程序，代码段是必须的，但数据段不一定需要。

在 i8086 中选择菜单栏上的文件 → 新建项目或者使用快捷键 Ctrl+N，在 C:\sysclassfiles\interface\interface\Ex_1\asm 文件夹中创建一个新工程 Ex_1.asm。对应 2.4.1 中的 I/O 地址空间，大家利用上面的模版，编写代码如下：

```
;读 24 位拨码开关输出至 24 位 LED
CODE SEGMENT
   ASSUME CS:CODE
START:
   IN  AX, 10H    ;读拨码开关低 16 位
   OUT 0H,AX      ;输出至 LED 低 16 位
   IN  AX,12H     ;读拨码开关高 8 位
   OUT 2H,AX      ;输出至 LED 高 8 位
   JMP START
CODE  ENDS
      END START
```

完成读取拨码开关输出至 LED 的功能。

（3）　汇编与链接

点击按钮或按 Shift+F6 键汇编和链接刚输入的汇编程序。

当上述过程顺利完成时，会生产一个 EX_1.EXE，如图 5-43 所示。

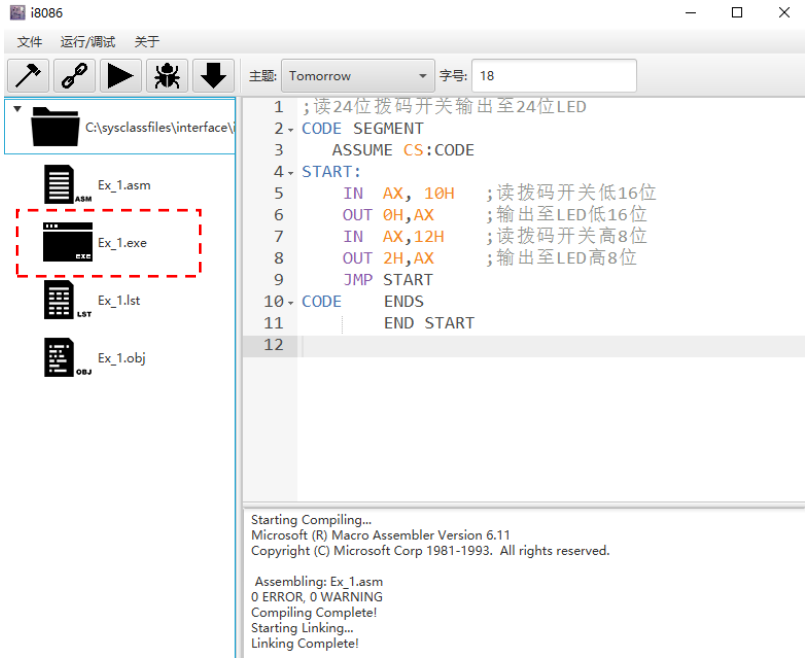


图 5-43 生成 Ex_1.exe 文件

（4）　将EXE文件转换成COE文件和TXT文件

点击按钮或按 Shift+F10 键，将生成的 EX_1.EXE 文件转换为 EX_1.COE 和 EX_1.TXT 文件，如所示。

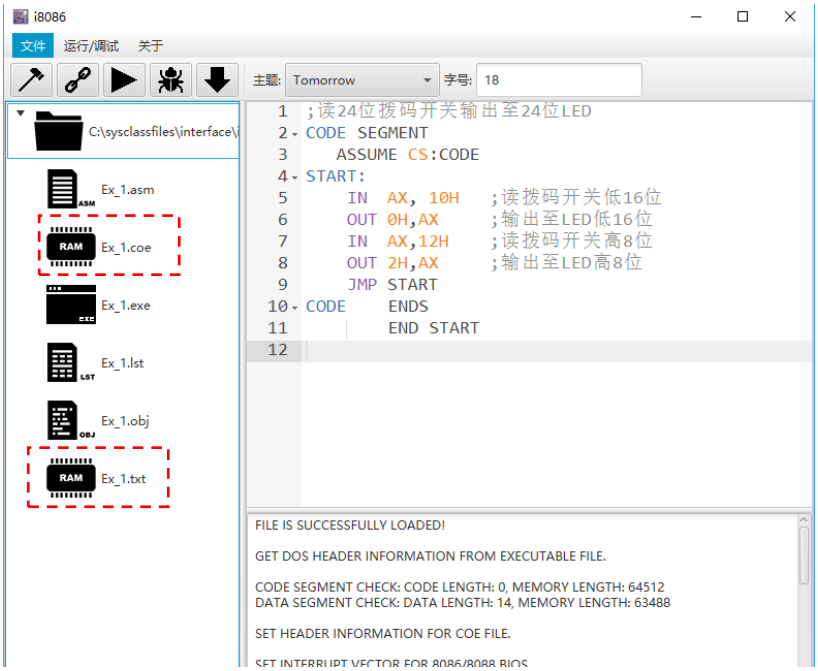


图 5-44 生成 Ex_1.COE 和 Ex_1.TXT 文件

（5）　加载coe

双击 C:/sysclassfiles/interface /interface /Ex_1/Ex_1/ Ex_1.xpr，打开最开始建立的工程，找到 Design Sources 工程文件夹下 Coefficient Files 子文件夹下的 init.coe（如图 5-45 所示），点击打开。将 C:\sysclassfiles\interface\interface\Ex_1\asm\EX_1.coe 里面的内容全部复制到过去，即重写 init.coe。

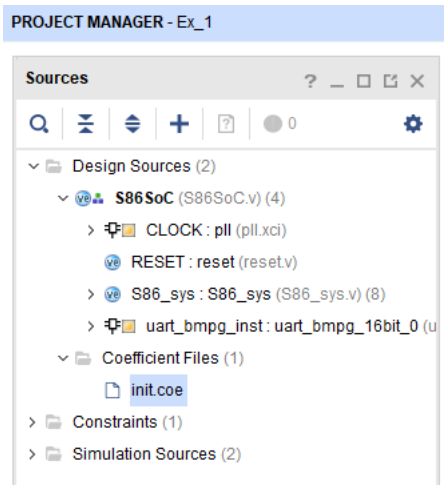


图 5-45 init.coe 的位置

重写完并保存后，双击图 5-46 中高亮部分，打开已建的内存 IP 核配置界面，选择 Other Options 选项，用 Browse 选择 C:\sysclassfiles\interface\interface\Ex_1\rtl_up\initfile\init.coe 重新加载 init.coe，如图 5-32 所示。

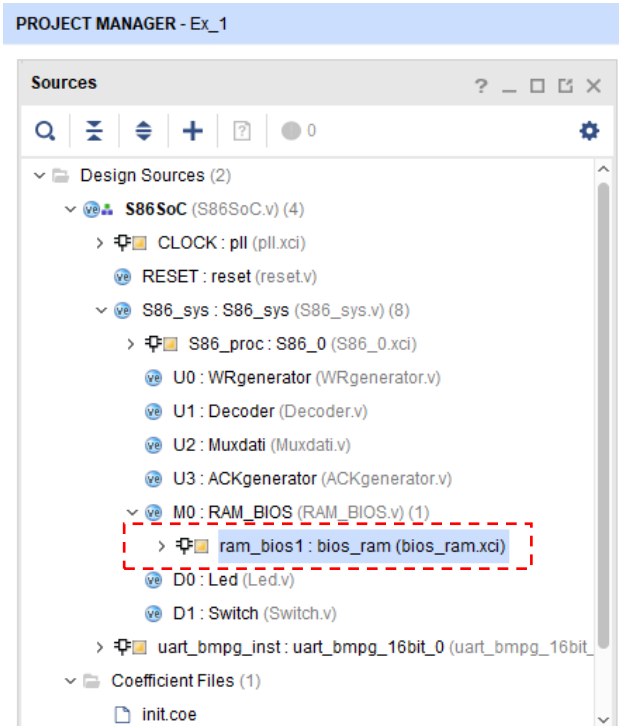


图 5-46 内存 IP 核的位置

加载完后点击 OK，点击 Genrate，同实验步骤 1 中的 block ramIP 核的添加。

（6）　仿真

完成上述步骤后，我们将进行仿真，作为下板前的验证。

右键点击 Simulation　Sources，如图 5-47 所示。

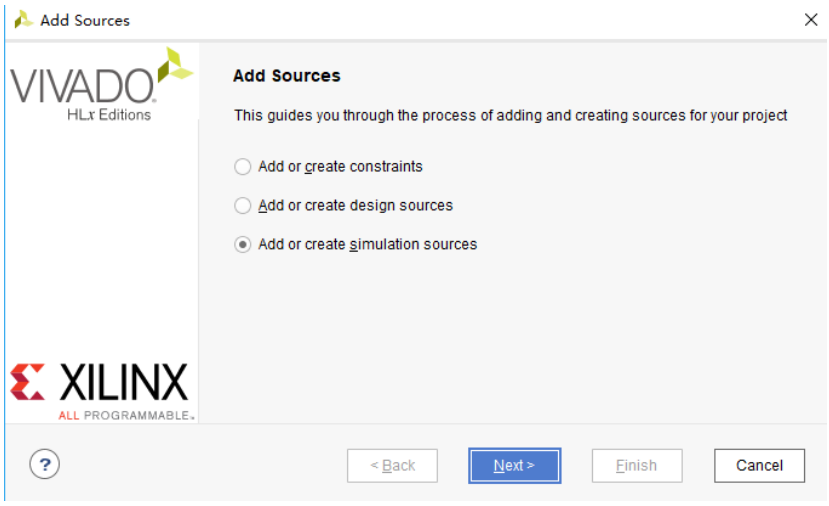


图 5-47 添加仿真文件

点击 Next，点击 Add Files，将 C:\sysclassfiles\interface\interface\Ex_1\sim\Ex_1.wcfg 和 C:\sysclassfiles\interface\interface\Ex_1\sim\S86_sim.v 两个文件添加入工程如图 5-48 所示。

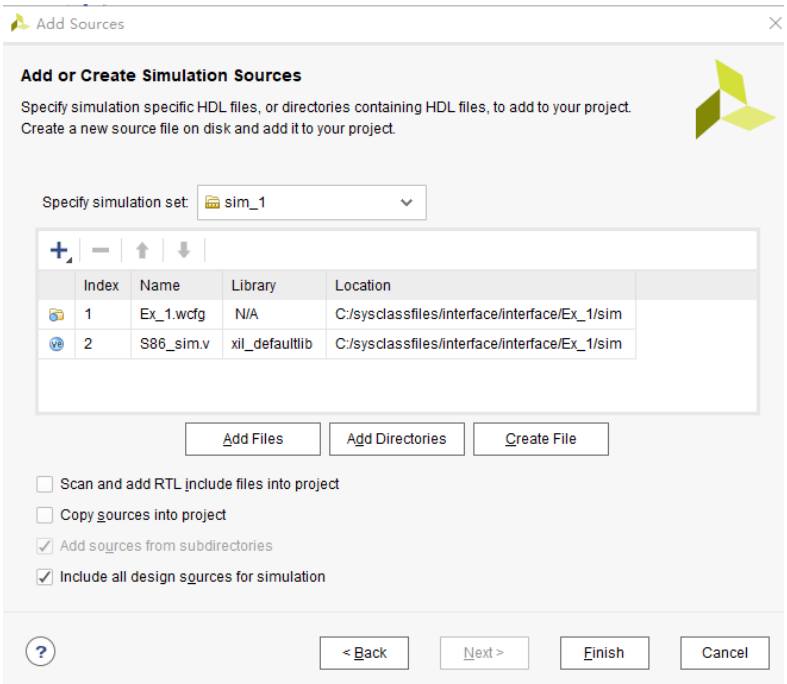


图 5-48 选择文件

点击 Finish，可以看到文件被加入到工程中，如图 5-49 所示。

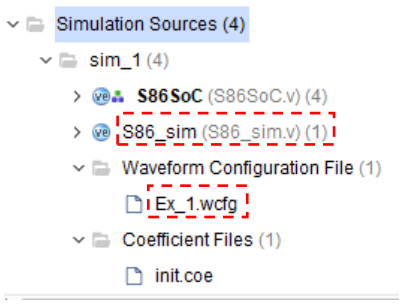


图 5-49 文件加入工程

右键点击 S　86_sim，在弹出的菜单中点击 Set as Top，将 S86_sim 设置为顶层文件。如所示。

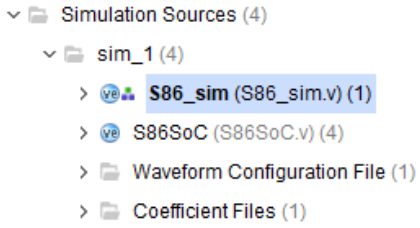
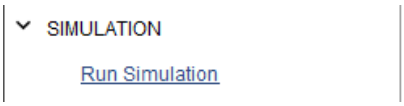
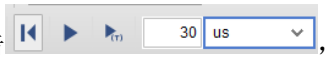


图 5-50 S86_sim 成为顶层文件后被加粗

点击 Simulation 下的 Run Simulation 

，点击 Run Behavior Simulation 进行行为仿真，点击 

，开始仿真。

可以看到当 swcsn 和 ior_n 同时有效时，当 wb_adr_i 为 0 时，表示 switch_低 16 位的值被写入 switch_out，如图 5-50 所示。

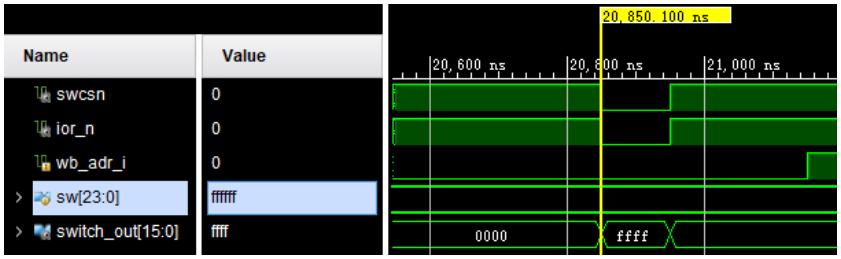


图 5-51 SW 仿真

而当 ledcsn 和 iow_n 同时有效时，当 wb_adr_i 为 0 时，表示将前面读取的低 16 位的值被写入 led_的低 16 位，如图 5-51 所示。

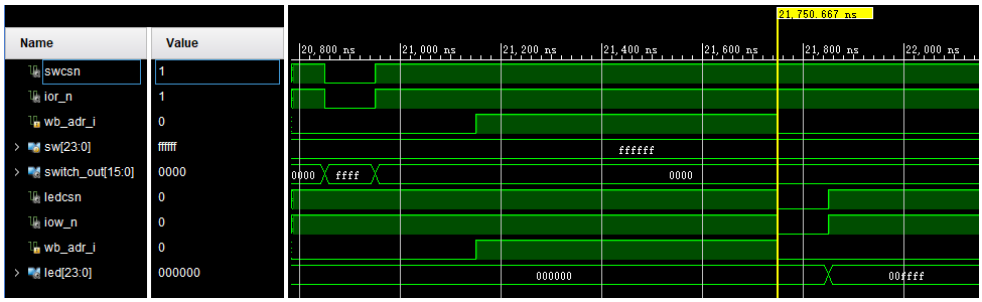


图 5-52 LED 仿真

通过仿真可以看出设计符合要求。

（7）　生成比特流文件和下载

由于 xdc 文件已经在第一步中加入，管脚分配已完成，只需直接点击 Promgram and Debug 下的 Generate Bitstream 或者工具条上的  按钮。即可生成比特流文件，如图 5-52 所示。

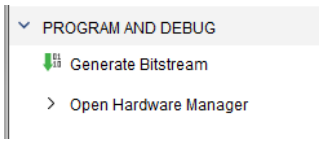


图 5-53 生成比特流文件

比特流生成后会出现图 5-53 所示的对话框

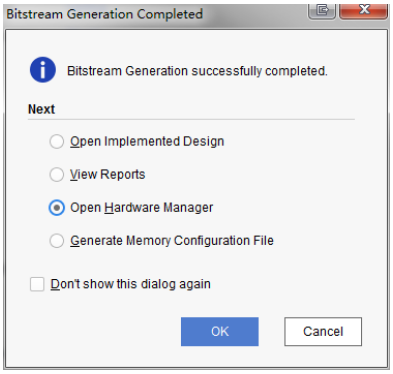


图 5-54 比特流生成完成

用 USB 下载线将 Minisys 板的 PROG UART（2016 版的 Minisys 板）或 Type C（2017 版的 Minisys 板）与 PC 机的 USB 相连，**打开 Minisys 板的电源**。按照所示选中 Open Hardware Manager。点击 **OK**。（在 Project Manager 中点击 Hardware Manager）。出现图 5-54 所示的界面。

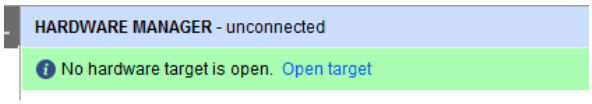


图 5-55 Hardware Manager

点击 Open target。在弹出的窗口中点击 Auto Connect。此时出现图 5-55 所示的滚动条。

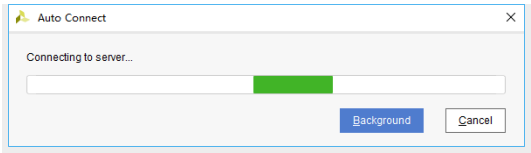


图 5-56 寻找硬件

硬件连接上后，出现如图 5-56 所示界面。

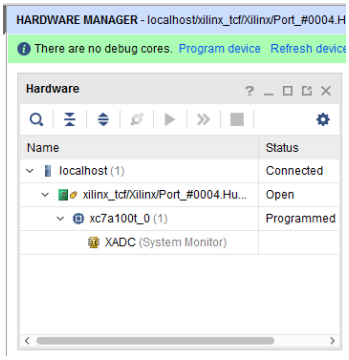


图 5-57 连接上硬件后

点击 Program devices。出现的 Program Device 窗口（如图 5-57）中点击 **Program**。

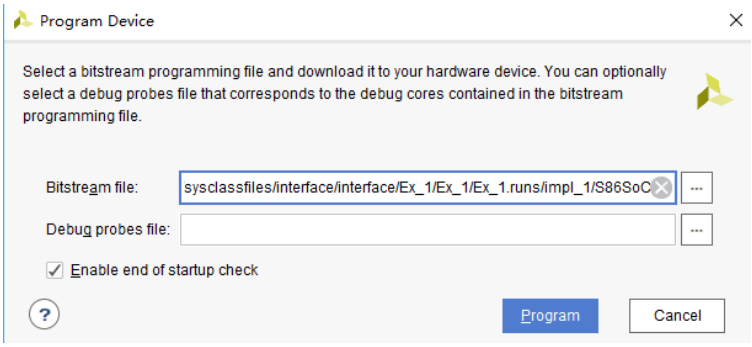


图 5-58 Program Device 窗口

出现如图 5-58 所示的正在下载的进度条。

图 5-59 下载进度条

（8）　实验现象

下载结束后，按一下 FPGA 复位按钮，会看到 Minisys 板上的 LED 灯会随着拨码开关的变化而变化。

（9）　总结

按照上述步骤，首先进行工具的安装，然后编写汇编程序，编译链接生成 exe 文件，接着将 exe 文件通过 exe2coe 转换成 coe 文件，将 coe 文件重新加载进内存，然后进行仿真，最后生成比特流文件，下载到板子。

注意，如果硬件没有更改，只是汇编程序做了修改，那么可以参考 4.1.1 中加载.txt 的方法，快速下载修改后的程序。

6. 思考与研讨

以下是 LED 输出模块：

```
module Led(
    // Wishbone slave interface
    input         wb_clk_i,
    input         wb_rst_i,
    input  [15:0] wb_dat_i,
    input         wb_adr_i,
    input         IOW_N,
    input         CS_N,
    output  reg [23:0] ledout
    );
    always @ (posedge wb_clk_i or posedge wb_rst_i) begin
        if(wb_rst_i) begin
```

```
            ledout <= 0;
        end else if ((!CS_N)&(!IOW_N)&(!wb_adr_i)) begin
            ledout[15:0] <= wb_dat_i;
        end else if((!CS_N)&(!IOW_N)&(wb_adr_i)) begin
            ledout[23:8] <= wb_dat_i;
        end
    end
endmodule
```

请问：output　reg [23:0] ledout 为何需要定义成寄存器类型？由此总结对于简单的接口信号输出最基本电路应该包含什么？

5.2　数码管实验

5.2.1　1 位 7 段数码管的设计

1. 实验目的

熟悉 1 位 7 段数码管的控制。

2. 实验内容及要求

使用 Verilog HDL 实现一个 1 位 7 段数码管的编码器 hexseg，可以在数码管上显示 1 位 16 进制数，下载到板子上进行验证。信号对应如下：输入 hex[3]-hex[0]对应 SW[3]-SW[0]，表明输入的 4 位二进制数，输出 segs[6]-segs[0]分别对应数码管的 CA,CB,CC,CD,CE,CF,CG 段，输入 en 接 SW23, 输出 an 接板上数码管使能引脚 A7~A0，其中 en 连接到 an[0]来控制第一个数码管的显示和不显示。 请根据 0 中图 1-8 自行列出各个数字的 7 段数码管真值表，并按照真值表设计 7 段数码管编码器。要注意，数码管哪一段亮，该段对应的信号要为低电平。

3. 实验预习

1) 复习本讲义 0 和第 2 章的内容。
2) 理解数码管七段码与 0~9，a~f 数字与字幕显示的关系。
3) 根据 2）的理解，完成表 5-3 所示的数码管显示 0~9，a~f 数字与字母的真值表。

表 5-3 7 段数码管真值表

输　入（hex[3:0]）	输　出（segs[6:0]=abcdefg）
无显示	1 1 1 1 1 1 1
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
b	
C	
d	
E	
F	

4. 实验步骤

1） 创建一个项目

新建一个项目，名称为 Ex_2_hexseg。

2） 添加源代码文件和引脚约束文件

将 C:\sysclassfiles\interface\Ex_2\Ex_2_hexseg 中 hexseg.v 和 hexseg.xdc 加入到工程。

3） 硬件代码设计

打开 hexseg.v，请完善下面设计中的???部分：

```
module hexseg(
        input [3:0] hex,
        input en,
        output [7:0] an,
        output reg [6:0] segs
        );
        assign an[7:0] = {7'b1111111, en};
        always @(*)
           case(hex)
                  // abc_defg
              4'h0: segs = 7'b000_0001;
                ???
              default:
             segs = 7'b111_1111;
           endcase
    endmodule
```

4） 仿真

先添加 C:\sysclassfiles\interface\Ex_2\Ex_2_hexseg\Ex_2_hexseg_sim.v 文件和 C:\sysclassfiles\interface\Ex_2\Ex_2_hexseg\Ex_2_hexseg_sim_behav.wcfg 文件，然后进行仿真，可以看到如图 5-59 所示的仿真结果，其中 hex 对应的 segs 要与真值表一致，an[0]与 en 值相同。

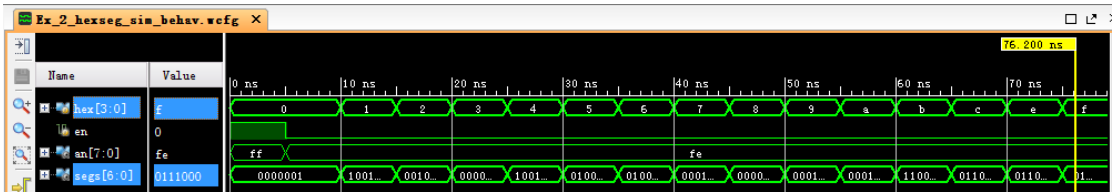


图 5-60 仿真结果图

5） 综合、实现、生成比特流文件和下载

请参见 5.1 中的相关步骤。

5. 实验现象

SW23 置高，改变 SW3-SW0，数码管不亮。

SW23 置低，改变 SW3-SW0，数码管显示对应的值。

5.2.2　4 位 7 段数码管的设计

1. 实验目的

进一步熟悉接口设计，熟悉 4 位 7 段数码管的控制。

2. 实验内容及要求

1) 将实验一中 1 位数码管控制扩张到 4 位数码管显示。要求：4 位数码管可显示不同内容；4 位数码管显示平稳、不闪烁，也不产生笔划缺失；
2) 设计电路，根据表 2-5 地址分配将（1）中数码管显示控制用适当的接口方式接到 S86 系统中。要求：为程序提供以下两个端口：

（1）低 4 位数码管数据端口（**00020H**）

数据寄存器格式如表 5-4 所示（复位时默认为全 0）：

表 5-4 低 4 位数码管数据寄存器

位数	15:12	11:8	7:4	3:0
意义	A3 数据的输入	A2 数据的输入	A1 数据的输入	A0 数据的输入

（2）低 4 位数码管控制端口（**00024H**）

控制寄存器格式如**错误!未找到引用源。**所示（其中高电平表示亮）：

表 5-5 低 4 位数码管控制寄存器

位数	15:4	3	2	1	0
意义	保留	A3 的亮灭控制	A2 的亮灭控制	A1 的亮灭控制	A0 的亮灭控制

当某位为 1 数码管显示，位 0，相应数码管不显示。比如如果 4 位数码管显示 63，则只有低两位数码管显示，高两位不显示，因此 00024H 端口需要写 03H。

3) 编写汇编程序读取拨码开关将低 16 位 SW 按照 4 位一组用数码管进行显示。要求：本实验必须以汇编程序完成，不得以纯电路实现。

3. 实验预习

1) 复习本讲义 1.4.4 中有关多位数码管的内容和 2.4.1 中有关 I/O 地址的分配。
2) 理解多位七段数码管显示的原理。
3) 理解 00024H 端口的作用。

4. 实验原理

该实验中每个数码管显示的数不同，但是数码管的段（segs）是公用的。所以，同一时间只点亮一个数码管，并将该数码管对应段码输入。所以根据人眼的视觉停留，本实验中需要一个计数器产生大约 2ms 一次的数码管切换触发信号，依次点亮数码管，并灭掉其他数码管。

由于一共是 4 位的数码管，每个数码管对应的 hex 输入为 4 位，所以数据寄存器设置为 16 位，并通过数据端口进行访问。4 位数码管每一个的亮灭通过控制寄存器进行控制，一共需要 4 位。控制寄存器通过控制端口进行访问。

故本实验中相应接口信号定义如表 5-6 所示：

表 5-6 数码管模块接口信号定义

信号名称	位宽	类型	意义
wb_clk_i	1	输入	时钟信号
wb_rst_i	1	输入	复位信号
wb_adr_i	2	输入	地址信号
wb_dat_i	16	输入	数据信号
IOW_N	1	输入	I/O 写信号

CS_N	1	输入	片选信号
an	8	输出	数码管选通信号
segs	8	输出	数码管段信号

各个信号具体功能描述如下：

1、wb_clk_i：时钟信号。用以分频产生 2ms 时钟来控制数码管切换。

2、wb_rst_i：复位信号。用以进行相关逻辑的复位。

3、wb_adr_i：地址信号。用以低位地址和片选配合选择访问的端口。

4、wb_dat_i：数据信号（输入）。来源于 S86，写端口时，写入的数据。

5、IOW_N：I/O 写信号。低电平有效。

6、CS_N：片选信号。低电平有效，表示对数码管进行访问。

7、an：数码管选通信号。相应位低电平表示点亮该数码管。

8、segs：数码管段信号。用以控制数码管显示。

5. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface\Ex_2\Ex_2_hexseg4\ Ex_2_hexseg4.xpr，打开工程。

2） 完成模块设计

点击工程目录中的 Digit.v。根据文件中的提示，完成相关设计。此处也可以使用自己设计的模块，但是接口信号要保持一致。

3） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

4） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

5） 应答信号发生器修改

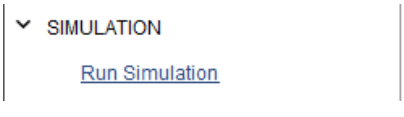
在应答信号发生器上将片选信号和相应逻辑加上。

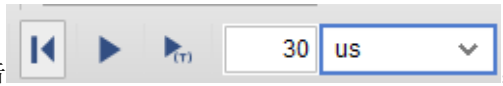
6） 写S86 汇编程序，完成读开关写数码管的操作

7） 编译、链接、转换变成coe加入工程文件中

8） 仿真

由于实际使用的时钟的时钟周期为 2ms，会导致仿真时间很长，所以先在设计文件中将时钟降低为 2μs，然后再进行仿真。

点击 Simulation 下的 Run Simulation 

，点击 Run Behavior Simulation 进行行为仿真，点击 

，开始仿真。

可以看到如图 5-60 所示的仿真结果。写控制寄存器和数据寄存器逻辑正确，且 segs 和 an 相对应。

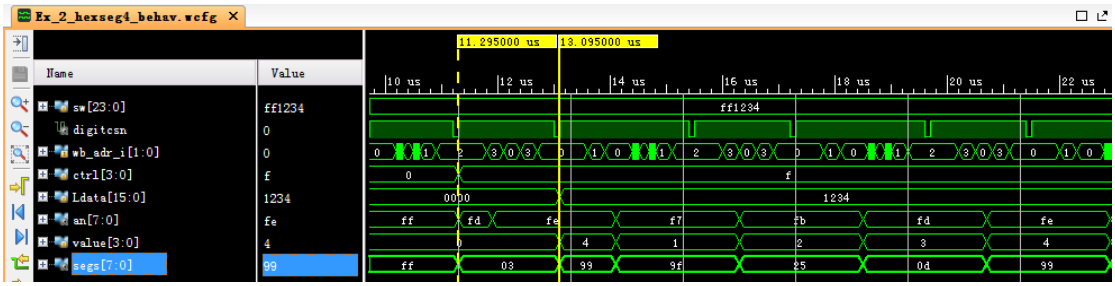


图 5-61 仿真结果图

9） 综合、实现、产生比特流文件、下载和运行

6. 实验现象

数码管 A3~A0 依次显示 SW15-SW12，SW11-SW8

7. 思考与研讨

将 5.2.2 扩展为 8 位数码管，提示：请考虑增加 1 个数据端口（**00022H**），并将控制端口（**00024H**）扩展到 8 位，注意表 2-5 地址分配。

5.3　4×4 键盘的实验

1. 实验目的

进一步熟悉对数码管控制，熟悉 4×4 键盘的控制。

2. 实验内容及要求

1) 实现 4×4 键盘扫描电路。要求：能正确检测到 1 键按下并扫描该键位置；具有去除抖动功能；按图 5-61 所示返回按键编号。

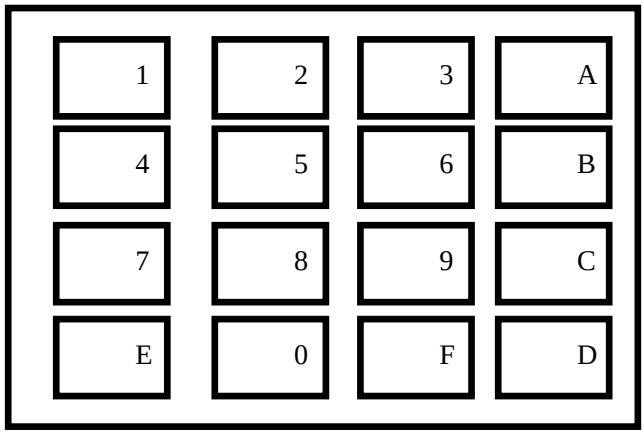


图 5-62 4×4 键盘示意图

2) 实现键盘接口电路。要求按照表 5-7 格式形成 4×4 键盘数据端口（**00030H**）数据寄存器的输入（不初始化默认为全 0）：

表 5-7 4×4 键盘数据寄存器

位数	15: 8	4	3:0
意义	保留	键值有效标记（高电平有效）	键值

3) 完成键盘到 S86 系统的集成。

4) 编写汇编程序，完成从键盘中读键在数码管 A1 和数码管 A0 交替显示。

3. 实验预习

1) 复习本讲义 1.4.3 中有关 4×4 键盘的内容和 2.4.1 中有关 I/O 地址的分配。

2) 理解 4×4 键盘的扫描原理。

3) 编写好相关程序。

4. 实验原理

如图 1-5 所示，非编码键盘的扫描方法大致如下：让所有行线全为 0，读出所有的列线状态，如果有列线为 0，则说明有键按下；然后从第 0 行开始，每扫描一行，令该行所对应的行线为 0，其余行线为 1。读入列线状态，如果有一列为 0，则该行该列交叉处的键被按下；如果所有列都为 1，则行号加 1，顺序扫描下一行。键号从左上角开始为 0 号，从左向右、从上到下依次编号，右下脚的编号为 15。

由于受制造工艺等影响，当按键按下时，在触点即将接触到完全接触这段时间里，键盘的通断状态很可能已经改变了多次，即在这段时间里，键盘输入了多次逻辑 0 和 1，也就是输入处于失控状态，这种情况通常称为按键抖动。如果这些输入被系统响应，则系统暂时也将处于失控状态，这是我们要尽量避免的。所以，需要对键盘进行消抖处理。对于 S86 系统，我们希望按下一次按键只产生一次有效键值，假定人常规按下按键到其弹起的时间大约为 200ms。在按键被按下后的 200ms 中键盘模块只提供一次有效的键值，也就是说按键按下后的 200ms 内 S86 访问键盘模块只能获取一次有效键值。这样在汇编编程上直接读取 4×4 键盘的数据端口（不用通过软件延时）就能和我们实际按键保持一致了。

为了实现上述的去抖电路，需要设计状态机来记录有效键值被访问的状态。所设计的状态机的状态转换图如图 5-62 所示。

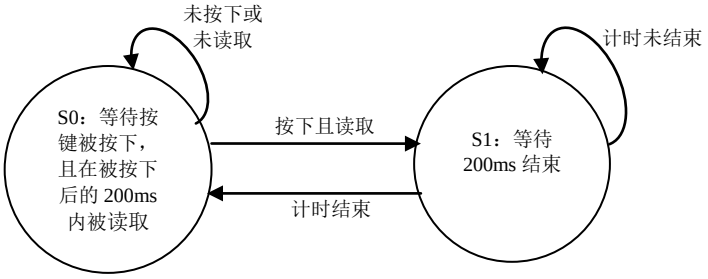


图 5-63 键盘消抖状态转换图

开始时，状态机处于 S0 状态，该状态判断是否有按键按下且按下后的 200ms 内是否通过程序进行访问，当按键按下且被访问时，将键值有效标志置 1，表示当前键值有效，并转入 S1。S1 状态中清除有效标志，并且等待当前的计数结束，当结束时转入 S0。该状态机的可行性有一个前提条件：**即 4×4 键盘模块的片选信号保持时间不能超过状态机的时钟周期，否则读取的有效标志依旧为 0**。所以我们选用的状态机时钟直接来源 S86 系统时钟即可。

故本实验中相应接口信号定义如表 5-8 所示：

表 5-8 4×4 键盘模块接口信号定义

信号名称	位宽	类型	意义
wb_clk_i	1	输入	时钟信号
wb_rst_i	1	输入	复位信号
IOR_N	1	输入	I/O 读信号
CS_N	1	输入	片选信号
wb_dat_o	16	输出	数据信号
row	4	输入	4×4 键盘行信号
col	4	输出	4×4 键盘列信号

各个信号具体功能描述如下：

1、wb_clk_i：时钟信号。用以分频产生 200ms 计时和状态机时钟。

2、wb_rst_i：复位信号。用以进行相关逻辑的复位。

3、IOR_N：I/O 读信号。低电平有效。

4、CS_N：片选信号。低电平有效，表示对 4×4 键盘进行访问。

5、wb_dat_o：数据信号（输出）。输出至输入数据多路选择器。

6、row：4×4 键盘行信号。与 col 配合用作行列扫描。

7、col：4×4 键盘列+信号。与 row 配合用作行列扫描。

注意：此处对低位地址进行了简化，因为该模块只有一个数据寄存器，故在可区分其他外设的前提下，不需要传入低位地址，但是汇编设计上请严格按照约定访问 **00030H 端口**。

5. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface\Ex_3\Ex_3_keypad\ Ex_3_keypad.xpr，打开工程。

2） 完成模块设计

点击工程目录中的 Keypad.v。根据文件中的提示，完成相关设计。

3） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

4） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

5） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

6） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑

7） 写 S86 汇编程序，完成从键盘中读键在数码管 A0 和 A1 交替显示功能

8） 编译、链接、转换变成coe加入工程文件中

9） 仿真

由于计数的周期为 200ms，会导致仿真时间很长，所以先在设计文件中将时钟，然后在进行仿真，图 5-63 修改仿真时钟所示。

```
43    always @(posedge wb_clk_i) begin
44        if(counter_clk < 2000000) begin    //200ms
45    //    if(counter_clk < 200) begin    //仿真用时钟
```

图 5-64 修改仿真时钟

点击 Simulation 下的 Run Simulation，点击 Run Behavior Simulation 进行行为仿真，点击，开始仿真。

可以看到如图 5-64 所示的仿真结果。行列译码结果正确，且一次按下只能读取一次有效键值。

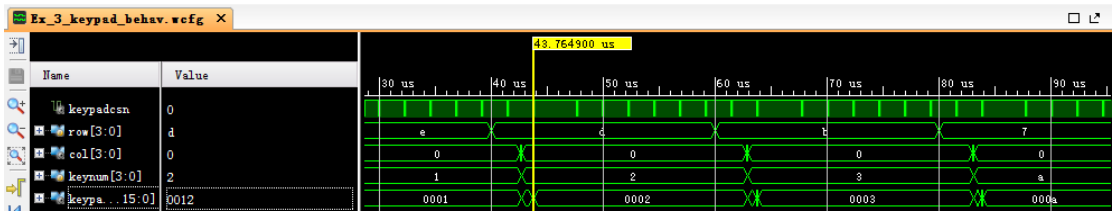


图 5-65 仿真结果图

10） 综合、实现、产生比特流文件、下载和运行

6. 实验现象

数码管 A0 和 A4 交替显示 4×4 键盘按下的键值。

7. 思考与研讨

1) 利用地址表技术使用读入的键值完成调用不同程序段的功能该如何编写程序
2) 考虑如果硬件只提供行输出、列输入信号由软件进行键盘扫描和去抖该如何实现。
3) 考虑将工程中提供的数码管 IP 核心替换为 5.2 中自己扩展设计的 8 位数码管模块。

5.4　计算器实验

1. 实验目的

加深对 S86 以及数码管和 4X4 键盘的理解，学会综合运用数码管和 4X4 键盘，完成一个简单综合应用系统的设计。

2. 实验内容及要求

1) 利用 5.2、5.3 设计的数码管模块和 4×4 键盘模块完成一个简单计算器的软、硬件设计（该两个模块也可采用系统资源提供的 C:\sysclassfiles\interface\IP_Peripheral 中的 CSE_Digit_1.0 和 CSE_Keypad_1.0　IP 核）。

2) 编写汇编程序完成简单的四则运算计算器功能：

（1）计算器布局如图 5-65 所示：

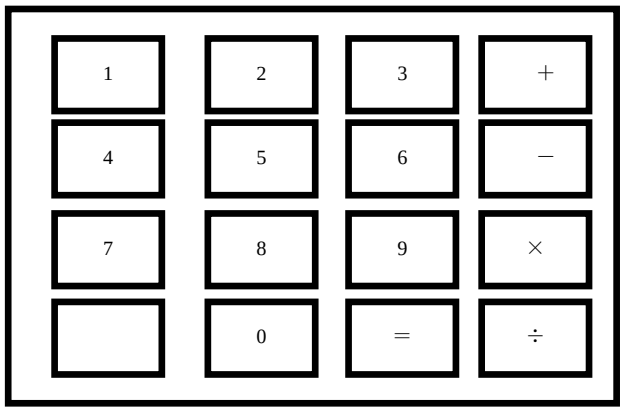


图 5-66 计算器布局

（2）运算规则如下：

a）加法：加数和被加数最多 4 位，仅 GLED0 亮表示加法。

b）减法：减数和被减数最多 4 位，不考虑结果为负，仅 GLED1 亮表示减法。

c）乘法：乘数和被乘数最多 4 位，仅 GLED2 亮表示乘法。

d）除法：除数和被除数最多 4 位，结果低四位输出商、高四位输出余数，仅 GLED3 亮表示除法。

e）不考虑连续运算。

f）运算错误，含减法结果为负，最左数码管 A0 显示为‘E’。

3. 实验预习

1) 复习 5.2、5.3 两节的内容。

2) 画好计算机程序的流程图

4. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface\Ex_4\Ex_4_calculator\ Ex_4_calculator.xpr，打开工程。

2） 将 5.2、5.3 设计模块或者将课程资源IP核加入工程

课程资源 IP 核为 C:\sysclassfiles\interface\IP_Peripheral 中的 CSE_Digit_1.0 和 CSE_Keypad_1.0。

3） 完成地址译码器端口连接

连接 I/O 片选信号、地址信号和 I/O 读写信号。

4） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

5） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

6） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑

7） 编写汇编程序，完成完成一个简单计算器

8） 编译、链接、转换变成coe加入工程文件中

9） 综合、实现、产生比特流文件、下载和运行

5. 实验现象

计算器符合要求能正确计算且能正确地显示运算错误。

6. 思考与研讨

请考虑数码管模块的改造以支持结果为负的减法运算输出

5.5　8 位类 8254 定时/计数器设计

1. 实验目的

根据 8254 内部结构，实现一个 8 位类 8254 单计数器。

2. 实验内容及要求

1) 根据 8254 内部结构，完成相关寄存器和端口设计。要求：必须实现方式一和方式三，其他方式可选做。
2) 编写 8254 初始化程序，完成 8254 初始化。
3) 为了简便操作，计数初值由 16 位简化为 8 位，SW23 和 RLED7 分别当做 GATE0 和 out0。
4) 编写汇编语言根据按键（SW0~SW11）完成表 5-9、表 5-10、表 5-11 指定功能：

表 5-9 写控制字

按键	有效值	功能
SW0	高电平	写控制字
SW1 （计数方式控制）	高电平	BCD 计数
	低电平	二进制计数
SW2~SW4 （工作方式控制）	X01	方式 1
	X11	方式 3

注：写方式字时，默认写 0 号计数器（即对应的方式控制字 D7D6 =00），方式控制字的 D5D4 只有 00（表示锁存计数器的当前计数值）和 01（表示写入时，只写低 8 位计数初值）两种取值。

表 5-10 写计数初始值

按键	有效值	功能
SW5	高电平	写初值
SW6~SW10 （写入的初值）	当 BCD 计数时(SW6~SW9) <=9	

注：8 位初值为了简化操作，只使用 5 位，这样可区分 BCD 计数和二进制计数。

表 5-11 控制显示

按键	有效值	功能
SW11	高电平	读回命令，锁存状态
	低电平	读取计数值

另外，低 4 位数码管（0 组）通过设计汇编程序实现根据 SW11 显示状态或者计数值。

3. 实验预习

1) 认真复习 8254 的原理与结构；
2) 注意实验要求中与真实 8254 的区别

4. 实验原理

由于本实验要设计的是类 8254，所以，我们先回顾 8254 的原理。

1） 8254 内部结构

8254 的内部结构框图如图 5-66 所示。它由与 CPU 的接口、内部控制电路和三个计数器组成。

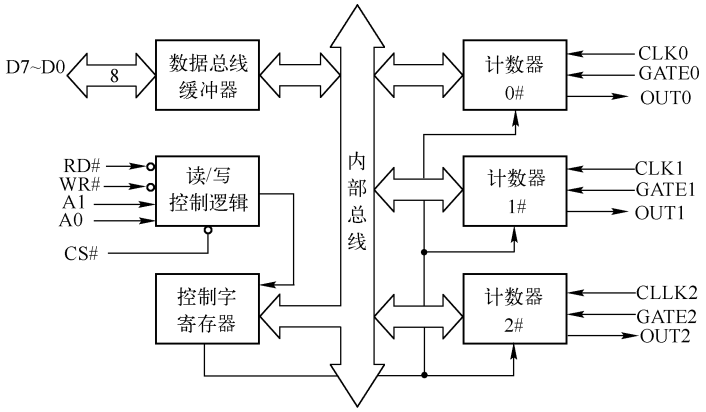


图 5-67 8254 内部结构框图

（1） 数据总线缓冲器

数据总线缓冲器是一个三态、双向 8 位寄存器，用于将 8254 与系统数据总线相连。数据总线缓冲器有 3 个基本功能：CPU 通过数据总线缓冲器向 8254 写入确定工作方式的命令字，向某一计数器写入计数初值，从某一计数器读取当前的计数值。

（2） 读/写控制逻辑

这是 8254 内部的控制电路，当片选信号 CS#=0 时，由 Al 和 A0（通常接系统地址线 Al 和 A0）信号选择内部寄存器，由读信号 RD#（通常接系统控制总线的 IOR#）和写信号 WR#（通常接 CPU 的 IOW#）完成对选定寄存器的读/写操作。

（3） 控制字寄存器

控制字寄存器是 8 位只写寄存器。初始化编程时，由 CPU 写入控制字，以决定计数器的工作方式；计数器工作过程中，可由 CPU 写入读出命令。

（4） 计数器

三个计数器相互间是完全独立的，但结构和功能完全相同，都由以下部件组成。

① 计数初值寄存器 CR（16 位）：用于存放计数初值（针对不同应用场合，又称定时常数或分频系数），其长度为 16 位，故最大计数值为 65536。计数初值寄存器的初值是和减 1 计数器的初值在初始化时同时一起装入的，计数初值寄存器的计数初值，在计数器计数过程中保持不变。故计数初值寄存器的作用是，在自动重装操作中为减 1 计数器提供计数初值，以便重复计数。所谓自动重装是指当减 1 计数器减 l 至零后，可以自动把计数初值寄存器的内容再装入减 1 寄存器，重新开始计数。当写入控制字时，将同时清除计数初值寄存器的内容。

② 计数器工作单元 CE（16 位）：用于进行减 1 计数操作，每来一个时钟脉冲，它就做减 1 运算，直至将计数初值减为零。CE 是 CPU 不能直接读/写的，需要修改其初值时，只能通过写入 CR 实现。

③ 输出锁存器 OL（16 位）：用于锁存减 1 计数器的内容，以供读出和查询。在计数过程中，OL 随 CE 的变化而变化。由于 CE 的内容是随输入时钟脉冲在不断改变的，为了读取这些不断变化的当前计数值，只有先把它送到 OL 加以锁存才能读出。经锁存后的 OL 内容将一直保持至 CPU 读出时为止。在 CPU 读出 OL 内容之后，OL 又跟随 CE 变化。

④ 状态寄存器：用于保存当前控制字寄存器的内容、输出的状态，以及 CR 内容是否已装入 CE 的指示状态，同样必须先锁存到状态锁存器，才允许 CPU 读取。

计数工作单元 CE 和计数初值寄存器 CR 及输出锁存器 OL 均为 16 位，而内部总线的宽度为 8 位，因此 CR 的写入和 OL 的读出都必须分两次进行。若在初始化时只写入 CR 的一个字节，则另一个字节的内容保持为零。

每个计数器对外有 3 个引脚：GATEi 为门控信号输入端，CLKi 为计数脉冲输入端，OUTi 为信号输出端。

初始化编程时，向计数初值寄存器写入的计数初值（只要不写入新的初值，该值始终保持不变），将自动送入 16 位计数器工作单元。当计数器允许计数时，每一个 CLKi 信号的下降沿使计数器工作单元减 1。当计数值减到某个规定数值时（取决于工作方式的设定），OUTi 端产生输出信号。

计数脉冲可以是有规律的时钟信号，也可以是随机脉冲信号。

计数初值 n 的计算公式如下：

$n=f_{CLKi}/f_{OUTi}$

其中，f_{CLKi} 是输入时钟脉冲的频率，f_{OUTi} 是输出波形的频率。

2） 外部引脚

8254 的外部引脚（见图 5-67）分为与系统总线连接的信号线和与其他设备连接的信号线。

● 面向系统总线的信号线有：

D7～D0——数据总线，为三态输出/输入线。用于将 8254 与系统数据总线相连，是 8254 与 CPU 接口数据线，供 CPU 向 8254 进行读/写数据、传送命令和状态信息。

CS#——片选线，为输入信号，低电平有效。当 CS#为低电平时，CPU 选中 8254，可以向 8254 进行读/写；当 CS#为高电平时，CPU 未选中 8254。CS#由 CPU 输出的地址码经译码产生。

RD#/WR#——读/写信号，为输入信号，低电平有效。它由 CPU 发出，用于对 8254 寄存器进行读/写操作。

A1 和 A0——地址线，这两根线一般分别接到系统地址总线的 A1 和 A0 上。当 CS#=0，8254 被选中时，A1 和 A0 用于选择 8254 内部寄存器，以便对它们进行读/写操作。8254 内部寄存器与地址线 A1 和 A0 的关系如表 5-12 所示。

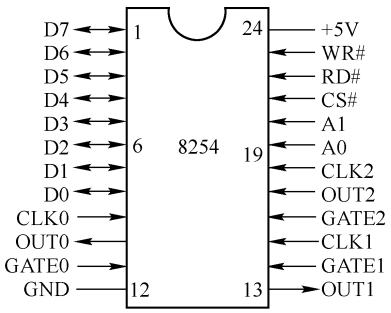


图 5-68 8254 外部引脚图

表 5-12 IA-32 微机中 8254 内部寄存器读/写操作

CS#	RD#	WR#	A1	A0	I/O 地址	操　　作
0	1	0	0	0	40H	计数初值写入 0#计数器
0	1	0	0	1	41H	计数初值写入 1#计数器
0	1	0	1	0	42H	计数初值写入 2#计数器
0	1	0	1	1	43H	向控制字寄存器写控制字
0	0	1	0	0	40H	读 0#计数器当前计数值
0	0	1	0	1	41H	读 1#计数器当前计数值
0	0	1	1	0	42H	读 2#计数器当前计数值
0	0	1	1	1	-	无操作
1	*	*	*	*	-	禁止使用
0	1	1	*	*	-	无操作

- 面向其他设备的信号线有：

CLK——计数器时钟信号，为输入信号。三个计数器各有一独立的时钟输入信号，时钟信号的作用是在 8254 进行定时或计数工作时，每输入 1 个时钟脉冲信号 CLK，便使计数值减 1。

GATE——计数器门控选通信号，为输入信号。三个计数器每一个都有自己的门控信号，作用是用来禁止、允许或开始计数过程的。对 8254 的 6 种不同工作方式，GATE 信号的控制作用不同。

OUT——计数器输出信号，三个独立计数器，每一个都有自己的计数器输出信号，其作用是计数器工作时，每来一个时钟，脉冲计数器减 l，当计数值减为 0 时，就在输出线上输出一个 OUT 信号（高或低由工作方式决定），以示定时或计数已到。这个信号可作为外部定时和计数控制信号接到 I/O 设备，用来启动某种操作（开/关或启/停）；也可作为定时和计数已到的状态信号供 CPU 检测；或作为中断请求信号使用

3） 工作方式

8254 的 3 个计数器均有 6 种工作方式，其主要区别在于输出波形、启动计数器的触发方式和计数过程中门控信号 GATE 对计数操作的影响不同。

工作于任何一种方式，都必须先写控制字至控制字寄存器，以选择所需方式，同时使所有逻辑电路复位，使 CR 内容清零，以及使 OUT 变为规定状态，再向 CR 写入计数初值；然后才能在 GATE 信号的控制下，以及在 CLK 脉冲的作用下进行计数。写入 CR 的计数初值范围，对二进制计数为 0001H～10000H，其中 10000H 用 0000H 代替；对十进制（BCD 码）计数为 0001～10000，其中 10000 用 0000 代替。

（1） 方式 0

方式 0 称为计数到零产生中断方式（Interrupt on Terminal Count）。它是典型的事件计数用法，当计数单元 CE 计至零时，OUT 信号由低变高，可作为中断请求信号。

8254 工作在方式 0 时，其工作波形如图 5-68 所示。

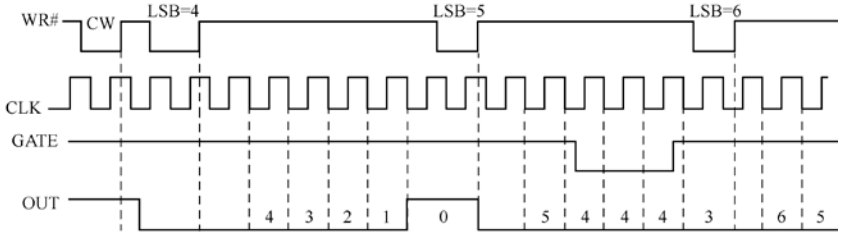


图 5-69 工作方式 0 波形图

方式 0 的工作特点是：

① 计数由软件启动，每次写入计数初值，只启动一次计数。当计数到零后，并不恢复计数初值，也不重新开始计数，OUT 端保持高电平（可做 8259A 的中断请求线）。只有再次写入计数值，OUT 变低后才开始新轮的计数。

② 8254 内部是在 CPU 写计数初值的 WR#信号的上升沿，将此值写入 CR。CR 内容并不立即装入 CE，而是在其后的下一个 CLK 脉冲下降沿才将 CR 内容装入 CE，开始计数，对该 CLK 脉冲并不计数。因此，若计数初值为 *n*，则必须在出现 *n*+1 个 CLK 脉冲之后，OUT 信号才变为高电平。这一特点在方式 l、方式 2、方式 4 和方式 5 中也同样具有。

③ 在计数过程中，如果 GATE=0，则暂停计数，直至 GATE 变 l 后再接着计数。

④ 在计数过程中可写入新的计数初值。从写入后的下一个时钟脉冲开始以新的初值计数。如果是 8 位计数，则在写入新的初值（仅低字节）后即按新值开始计数；如果是 16 位计数，则在写入第一字节后，计数器停止计数，在写入第二字节后，计数器才按新值开始计数。

（2） 方式 1

方式 1 称为硬件可重触发单稳（Hardware Retriggerable One-Shot）方式。该方式由外部门控脉冲（硬件）启动计数，相当于一个可编程的单稳态电路。其特点是：

① 写入控制字后，OUT 端输出高电平。写入计数初值后，OUT 端保持高电平，计数器由 GATE 的上升沿启动。GATE 启动之后，OUT 变为低电平，每来一个 CLK，计数器减 l；当计数值减到零时，OUT 输出高电平，从而在 OUT 端输出一个负脉冲。负脉冲宽度为计数初值乘以 CLK 脉冲周期。

② 方式 1 具有可重触发性，即允许多次触发。在计数器未减到零时，门控信号 GATE 又来一个正脉冲，计数初值将重新装入计数器，计数器从初值开始重新做减 l 计数。

③ 在计数过程中，程序员可装入新的计数初值，此时计数过程不受影响。只有当 GATE 再次出现上跳变后，计数器才能按新的计数初值做减 l 计数。

8254 工作在方式 l 时，其工作波形如图 5-69 所示。

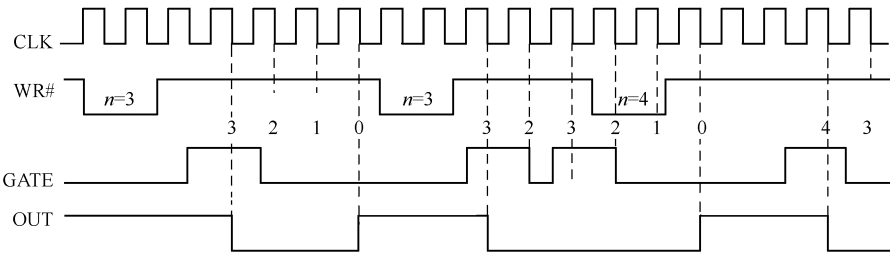


图 5-70 工作方式 1 波形图

（3） 方式 2

方式 2 为速率发生器（Rate Generator）方式，也叫 *n* 分频方式。方式 2 的特点是计数器有“初值自动重装”的功能，即计数值减到规定数值后，计数初值将会自动地重新装入计数器，所以能够输出固定频率的脉冲。其工作特点如下：

① 写入控制字后，OUT 输出为高电平。写入计数初值 *n* 后，如果 GATE 为高电平，则计数器开始做减 l 计数；当计数值减到 l 时，OUT 输出为低电平，维持一个 CLK 周期，又变为高电平，且计数初值 *n* 自动重装，计数器开始重新计数。如果 CLK 为周期性脉冲序列，则 OUT 端也输出周期性的负脉冲。负脉冲宽度为一个 CLK 周期，脉冲频率为 CLK 信号频率的 1/*n*，即为 CLK 的 *n* 分频信号。

② 如果在做减 l 计数的过程中，GATE 变低，则停止计数。GATE 的上升沿使计数器恢

复初值，并从初值开始做减 l 计数。

③ 在计数过程中，如果 GATE 为高电平，则程序员写入新的计数初值，不会影响正在进行的减 l 计数过程；只有计数器减到 l 之后，计数器才装入新的计数初值，并且按新的计数初值开始计数。

8254 工作在方式 2 时，其工作波形如图 5-70 所示。

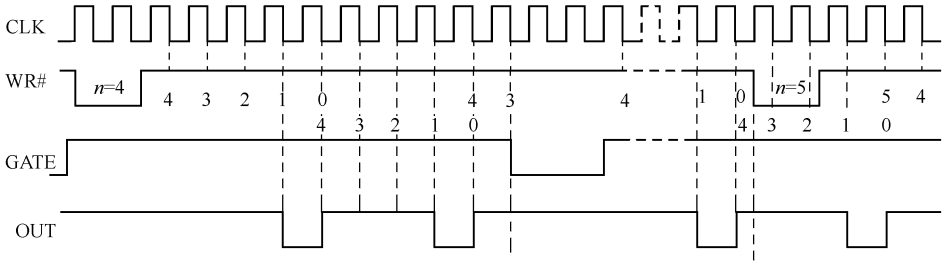


图 5-71 工作方式 2 波形图

（4） 方式 3

方式 3 称为方波发生器（Square Wave Output）方式。它的典型用法是作为波特率发生器。方式 3 有初值自动重装的功能。其工作特点如下：

① 写入控制字后，OUT 输出为高电平。写入计数初值 *n* 后，如果 GATE 为高电平，则在下一个 CLK 脉冲下降沿，计数器开始计数。

② 当计数初值为偶数的时候，每来一个 CLK 脉冲，计数值减 2；当计数值减到零的时候输出端改变极性，内部完成初值自动重装，继续计数。因此，输出端为 l:l 的方波，正脉冲和负脉冲的宽度均为 *n*/2 的 CLK 周期。

③ 如果计数初值为奇数，则在 OUT 变为高电平的瞬间，将 CR 内的计数初值减 1 后装入 CE，然后开始减 2 计数。在输出正脉冲期间，每一个 T_{clk} 使计数值减 2，当计数值减到 -2 时，输出端变成低电平，内部完成初值重装，重装的初值为编程时写入的初值减 l；在输出负脉冲期间，每一个 T_{clk} 使计数值减 2，当计数值减到零时，输出端变成高电平，内部完成初值重装，重装的初值也为编程时写入的初值减 l。输出的正脉冲宽度为 $T_{clk}(n+1)/2$，输出的负脉冲宽度为 $T_{clk}(n-1)/2$。

④ 计数的过程中，GATE 变低，则停止计数。GATE 的上升沿使计数器恢复初值，并从初值开始计数。

⑤ 与方式 2 一样，在计数过程中，如果 GATE 为高电平，则程序员写入新的计数初值，不会影响正在进行的计数过程；只有当前计数操作周期结束之后，计数器才装入新的计数初值，并且按新的计数初值开始计数。

8254 工作在方式 3 时，其工作波形如图 5-71 所示。

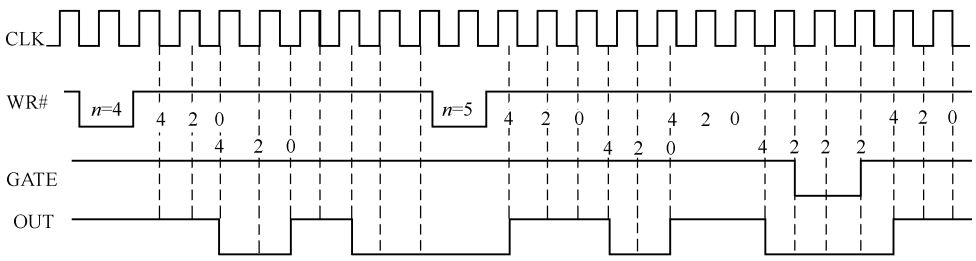


图 5-72 工作方式 3 波形图

（5）　方式 4

方式 4 为软件触发选通（Software Triggered Strobe）方式，与方式 0 十分相似。其工作特点是：

① 写入控制字后，OUT 输出高电平。若当前 GATE 为高电平，则写入计数初值后，开始做减 1 计数；当计数值减到零时，OUT 变低，在 OUT 端输出一个宽度等于一个 CLK 脉冲周期的负脉冲。

② 计数的过程中，GATE 变低，则停止计数。GATE 的上升沿使计数器恢复初值，并从初值开始做减 l 计数。

③ 在计数过程中，如果改变计数值，则按新的计数值重新开始计数。

8254 工作在方式 4 时，其工作波形如图 5-72 所示。

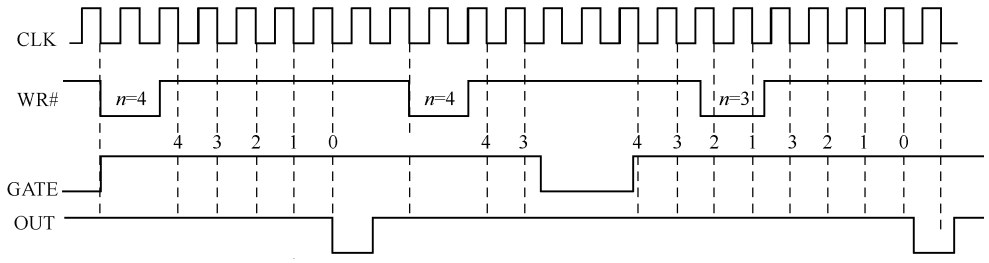


图 5-73 工作方式 4 波形图

（6）　方式 5

方式 5 称为硬件触发选通（Hardware Triggered Strobe）方式，与方式 1 十分相似。其工作特点是：

① 写入控制字之后，OUT 输出为高电平。写入计数初值后，只有在 GATE 端出现上跳变时，计数初值才能装入计数器，然后在 CLK 脉冲作用下，计数器做减 l 计数；当计数值减为 0 时，OUT 端输出一个 CLK 周期的负脉冲。

② 在计数过程中，若 GATE 端再次出现上跳变，则计数初值重新装入计数器，在 CLK 脉冲作用下，重新做减 l 计数。

③ 在计数过程中，如果改变计数初值，只要没有 GATE 上升沿触发，则不影响计数过程；若有，则立即按新的计数初值重新开始计数。

8254 工作在方式 5 时，其工作波形如图 5-73 所示。

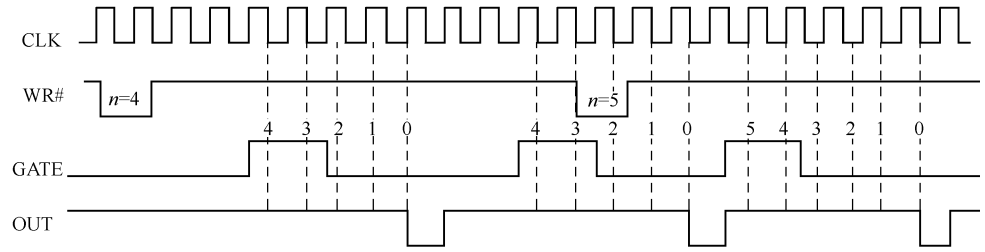


图 5-74 工作方式 5 波形图

方式 5 与方式 l 的区别是：方式 5 输出的单脉冲（负）宽度为一个 CLK 周期，而方式 1 输出的单脉冲（负）宽度为 *n* 倍的 CLK 周期（*n* 为计数初值）。

8254 的 6 种工作方式的比较表 5-12 所示。

表 5-13 8254 的 6 种工作方式的比较

		方式 0	方式 1	方式 2	方式 3	方式 4	方式 5
OUT 输出状态		写入控制字后变 0，计数结束变 1，并维持至重写控制字或计数初值	写入控制字后变 1，GATE 上升沿触发变 0，开始计数，计数结束变 1	写入控制字后变 1，计数到 1 变 0，维持一个 T_{clk} 变 1	写入控制字后变 0，装入计数初值且 GATE=1 则 OUT 变 1，计数到时变 0，重装计数初值继续计数，计数到时则反向	写入控制字后变 1，计数结束变 0，维持一个 T_{clk} 变 1	写入控制字后变 1，GATE 上升沿触发开始计数，计数结束输出一个宽度为 T_{clk} 的负脉冲
初值自动重装		无	无	计数到 0 重装	根据计数初值的奇偶分别重装	无	无
计数过程中改变计数初值		立即有效	GATE 触发后有效	计数到 1 或 GATE 触发后有效	计数结束或 GATE 触发后有效	立即有效	GATE 触发后有效
GATE 信号的作用	0	禁止计数	无影响	禁止计数	禁止计数	禁止计数	无影响
	下降沿	暂停计数	无影响	停止计数	停止计数	停止计数	无影响
	上升沿	继续计数	从初值开始重新计数	从初值开始重新计数	从初值开始重新计数	从初值开始重新计数	从初值开始重新计数
	1	允许计数	无影响	允许计数	允许计数	允许计数	无影响

4） 8254 控制字

（1） 方式控制字

方式控制字的格式如图 5-74 所示。

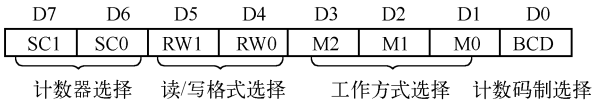


图 5-75 8254 的方式控制字格式

a) 计数器选择

D7D6=00，表示选择 0 号计数器。
D7D6=01，表示选择 l 号计数器。
D7D6=10，表示选择 2 号计数器。
D7D6=11，读出控制字的标志之一。

b) 读/写方式选择

D5D4=00，表示锁存计数器的当前计数值，以便读出检查。

D5D4=01，表示写入时，只写低 8 位计数初值 LSB（Least Significant Byte），高 8 位置零。读出时，只能读出低 8 位的当前计数值。

D5D4=10，表示写入时，只写高 8 位计数初值 MSB（Most Significant Byte），低 8 位置零。读出时，只能读出高 8 位的当前计数值。

D5D4=11，表示先读/写低 8 位计数值 LSB，后读/写高 8 位计数值 MSB。

c) 工作方式选择

D3D2Dl=000，计数器工作在方式 0。
D3D2Dl=001，计数器工作在方式 l。
D3D2Dl=X10，计数器工作在方式 2。

D3D2Dl=X11，计数器工作在方式 3。

D3D2Dl=100，计数器工作在方式 4。

D3D2D1=101，计数器工作在方式 5。

d) 数制选择

当 D0=0 时，计数初值被认为是二进制数，减 l 计数器按二进制规律减 l。初值范围是 0001H～10000H，其中 10000H（十进制的 65536）用 0000H 代替。

当 D0=l 时，计数初值被认为是二-十进制数，减 l 计数器按十进制规律减 l。初值范围是 0001H～10000H，其中 10000H（十进制的 10000）用 0000H 代替。

（2） 读出控制字

读出控制字的格式如图 5-75 8254 的读出控制字格式所示。

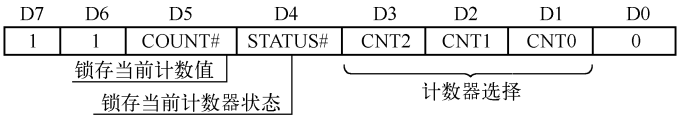


图 5-76 8254 的读出控制字格式

读出控制字 D7D6 必须为 11，D0 位必须为 0。D5=0 锁存计数值，以便 CPU 读取；D4=0 将状态信息锁存入状态寄存器。D3～Dl 为计数器选择，不论是锁存计数值还是锁存状态信息，都不影响计数。读出命令能同时锁存几个计数器的计数值/状态信息，当 CPU 读取某一计数器的计数值/状态信息时，该计数器自动解锁，但其他计数器不受影响。

（3） 状态字

状态字格式如图 5-76 所示。

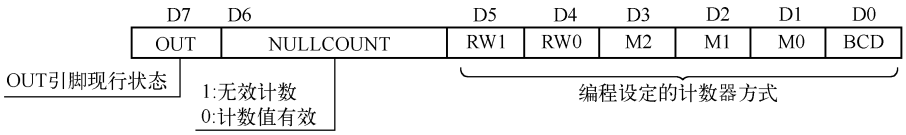


图 5-77 8254 的状态字格式

D5～D0 的意义与方式控制字的对应位意义相同。D7 表示 OUT 引脚的输出状态，D7=l 表示 OUT 引脚为高电平，D7=0 表示 OUT 引脚为低电平。D6 表示计数初值是否已装入减 l 计数器，D6=0 表示已装入，可以读取计数器。

5. 类 8254 模块接口信号

本实验中设计的类 8254 模块与实际 8254 有以下不同：

- 所设计的类 8254 模块只有一个计数器
- 所设计的类 8254 模块工作方式只有方式 1 和方式 3
- 所设计的类 8254 模块计数器所使用位数只有 8 位

本实验中相应接口信号定义如表 5-13 所示：

表 5-14 类 8254 模块接口信号定义

信号名称	位宽	类型	意义
clk0	1	输入	计数器 0 时钟信号
gate0	1	输入	计数器 0 门控选通信号
out0	1	输出	计数器 0 输出信号

CS_N	1	输入	片选信号
a	2	输入	地址信号
id	8	输入	数据信号（输入）
od	8	输出	数据信号（输出）
IOR_N	1	输入	I/O 读信号
IOW_N	1	输入	I/O 写信号

各个信号具体功能描述如下：

1、clk0：计数器 0 时钟信号。。

2、gate0：计数器 0 门控选通信号

3、out0：计数器 0 输出信号。

4、CS_N：片选信号。低电平有效，表示对 8254 进行访问。

5、a：地址信号。用以低位地址和片选配合选择访问的端口。（对低位地址进行了简化）

6、id：数据信号（输入）。输入 8254 模块的数据。

6、od：数据信号（输出）。从 8254 模块输出的数据。

7、IOR_N：I/O 读信号。低电平有效。

7、IOW_N：I/O 写信号。低电平有效。

6. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface \Ex_5\Ex_5_S8254\ Ex_5_S8254.xpr，打开工程。

2） 完成模块设计

点击工程目录中的 S_8254.v。根据文件中的提示，完成相关设计。

3） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

4） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

5） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

6） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑

7） 写S86 汇编程序，完成实验要求中的功能

8） 编译、链接、转换变成coe加入工程文件中

9） 仿真

由于输入计数器 0 的时钟的时钟周期为 1s，会导致仿真时间很长，所以先在 divclk.v 中将时钟调整为仿真用时钟，然后在进行仿真，如图 5-77 所示。

divclk.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3
4   module divclk(clk,rst,clk_sys);    //分频模块
5       input clk,rst;
6       output reg clk_sys;
7
8       reg [31:0] div_counter;
9       always @(posedge clk or posedge rst)     //时钟上升沿
10        if(rst) begin
11            clk_sys <= 0;
12            div_counter <= 0;
13        end
14  //      else if(div_counter >= 5000000) begin
15        else if(div_counter >= 20) begin//仿真用时钟
```

图 5-78 修改仿真时钟

添加仿真 coe，为了统一仿真标准，请先将 C:\sysclassfiles\interface \Ex_5\ sim.coe 加载在内存中，然后再进行仿真。

点击 Simulation 下的 Run Simulation，点击 Run Behavior Simulation 进行行为仿真，点击，开始仿真。

可以看到如图 5-78 和图 5-79 所示的仿真结果。方式寄存器中的值为 16h，初值为 1f，工作在方式 3 下，out0 高电平时计数到 0 翻转，计数值重装；out0 低电平时计数到 2 翻转写，计数值重装。OL 能够锁存 CE 的值，且能够准确读出 OL 的内容。

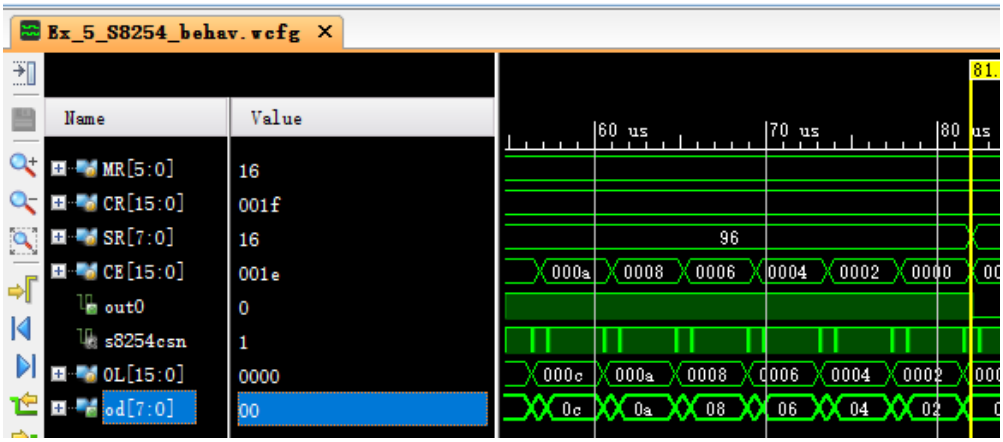


图 5-79 仿真结果图

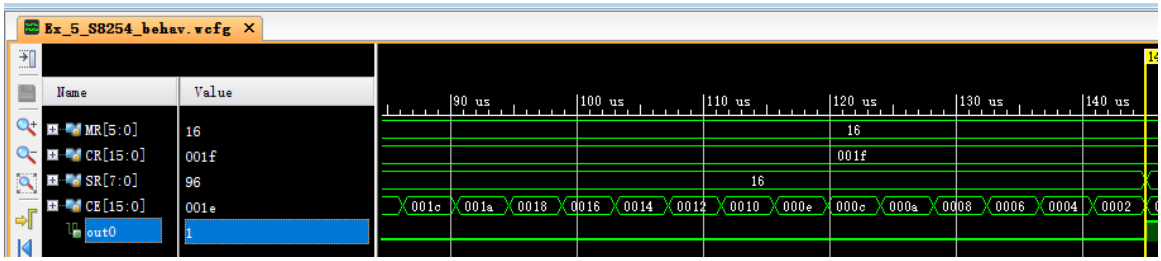


图 5-80 仿真结果图

10）　综合、实现、产生比特流文件、下载和运行

7. 实验现象

要求能满足实验要求中的所有功能。

8. 思考与研讨

1) 考虑实现 8254 的其他工作方式。
2) 考虑将 8 位计数器扩展到 16 位。
3) 考虑将单计数器扩展到 3 个计数器。
4) 用课程资源包中 C:\sysclassfiles\interface\IP_Peripheral 中 CSE_S_8254_1.0 IP 核，使用级联方式完成周期为 5S，占空比为 3：5 的方波发生器。

5.6　可编程中断控制器（PIC）设计

1. 实验目的

设计一个符合 S86 时序控制要求的可编程中断控制器并学会中断处理程序编写。

2. 实验内容及要求

1)　完成一个符合 S86 时序控制要求的可编程中断控制器。要求具有以下功能：

（1）可同时接受 8 个中断源（IR0~IR7），上升沿触发，不可级联。

（2）具有中断屏蔽功能。可屏蔽任何一个 IRi。

（3）采用固定优先级，IR0~IR7 优先级依次递减，PIC 具有优先级判断功能，中断嵌套采用普通全嵌套，只允许高优先级中断打断低优先级中断。

（4）能向 S86 发送中断。当 IRi 上升沿到来时置中断请求寄存器 IRR 相应位为 1，如果未被屏蔽且无高优先级中断正在服务，则通过 INT 发送可屏蔽中断请求信号。

（5）中断识别与中断向量号的输出功能，由于 S86 只有一个周期的中断响应号 INTA，故 PIC 在 INT 的上升沿保存中断向量号信息，并在 INTA#下降沿向 CPU 传送响应的中断向量号，清 IRR 相应位，置中断服务寄存器 ISR 相应位为 1，撤销 INT 信号。在 INTA#上升沿清除保存的中断向量号信息。

（6）中断结束功能。采用非自动结束功能，获得 CPU 中断结束命令后清除 ISR 相应位。

2)　实现中断屏蔽寄存器端口、中断请求寄存器端口和中断服务寄存器端口。实现初始化端口（ICW），同 8259ICW2。

3)　编写汇编程序完成中断相关程序设计，含 PIC 初始化、中断向量表初始化和中断处理程序编写。要求：接受 8 个中断请求，每个中断处理程序在 A0 数码管上显示 IRi 中的 i+1，如 IR0 显示‘1’，并用软件循环方法延迟 2S。

4)　PIC IR0~IR7 分为接 SW23~SW16。

5)　在 2 号中断中进行读取 SW7~SW0 设置 IMR。

3. 实验预习

由于 S86 与 8086 在中断应答信号上不同，因此 PIC 的设计在借鉴 8259 基础上做了适当的调整，清弄清楚 PIC 的原理。

4. 实验原理

1）PIC的内部结构

PIC 的内部结构如图 5-80 所示。

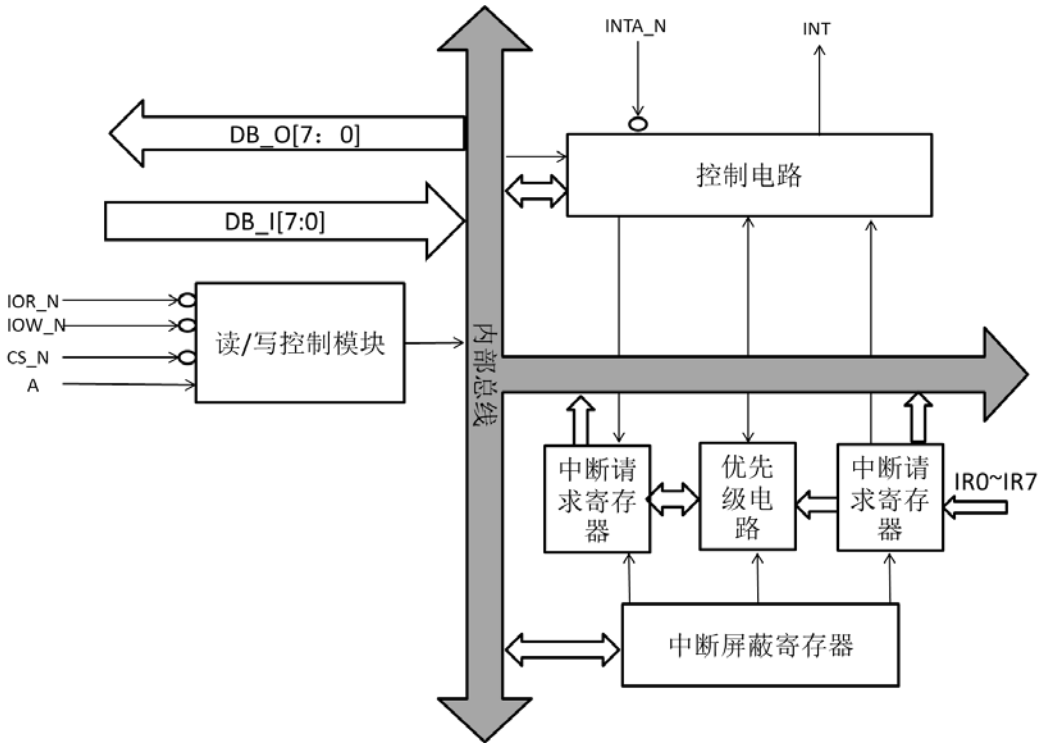


图 5-81 PIC 内部结构

（1） 中断请求寄存器（Interrupt Request Register，IRR）(8 位)

该寄存器存放对应引脚 IR7～IR0 的中断请求。当 IR*i* 中断请求信号有效时，IRR 中和 IR7～IR0 相应的 8 位（D7～D0）置“1”。具有锁存功能，其内容可以读 **00050H** 读出。

（2） 中断屏蔽寄存器（Interrupt Mask Register，IMR）(8 位)

该寄存器存放由程序设定的中断屏蔽字。IMR 中 8 位（D7～D0）对应 8 级中断屏蔽。如果希望哪级中断被屏蔽，则哪位就写“1”，即禁止该级提出中断请求；反之，写“0”，即允许该级提出中断请求（开放中断）。屏蔽操作由屏蔽命令 OCW1 执行。

（3） 优先级电路（Priority Resolver，PR）

该电路的功能是检查中断源中断请求的优先级并和“在服务寄存器”进行比较，确定是否让此中断请求送给 CPU。如果中断源的中断请求优先级比正在中断服务的中断优先级高，则“PR”就将此中断请求送给 CPU，并在中断响应时将它记入“在服务寄存器”的对应位中。如果中断源的中断请求的优先级等于或低于正在中断服务的中断优先级，则 PR 不为其将中断请求送给 CPU。

（4） 在服务寄存器（In Service Register，ISR）(8 位)

在中断响应之后，第一个中断回答 INTA#周期将允许中断请求（中断屏蔽寄存器相应位置“0”）的中断在相应位置“1”。所以，在服务寄存器是用来存放正在被服务的中断的，包括尚未服务完而中途被优先级更高的中断打断的中断。

（5） 读/写控制模块

该模块接收片选信号 CS_N、端口选择信号 A 和 I/O 读/写控制信号 IOR_N 及 IOW_N。PIC 需要两个端口地址，用地址线 A 来选择。

（6） 控制电路

它是 PIC 的内部控制器。它有一组用于寄存初始化命令字（ICW1～ICW2）的寄存器和

一组用于寄存操作命令字（OCW1～OCW2）的寄存器，以及相关的控制逻辑。这些命令字寄存器通过译码产生内部控制信号，可以根据中断请求、中断优先级的判别结果，通过引脚 INT 向 CPU 提出中断请求，通过引脚 INTA#接收中断响应信号。

2） PIC的工作方式

PIC 只支持边沿触发方式。以正跳变向 PIC 请求中断。正跳变后可一直维持高电平，不会再产生中断。

3） 屏蔽中断源的方式

PIC 采用常规屏蔽方式。利用操作命令字 OCW1 使中断屏蔽寄存器 IMR 中的一位或几位置“1”来屏蔽一个或几个中断源的中断请求。若要开放某一个中断源的中断请求，则将 IMR 中相应的位置“0”。

4） 优先级排队的方式

PIC 采用全嵌套方式。在此种方式下，中断优先级按 0～7 顺序进行排队，只允许中断级别高的中断源中断中断级别低的中断服务程序，而不能相反。这是 8259A 最常用的方式，8259A 复位后，即自动按此方式工作。

5） 结束中断的处理方式

PIC 使用非自动中断结束方式。在中断服务程序返回之前，必须发中断结束命令才会清除该中断服务程序所对应的 ISR 位。该中断结束命令为指定中断结束命令，即设置操作命令字 OCW2=01100$L_2L_1L_0$B，其中低 3 位 $L_2L_1L_0$ 的编码表示被指定要结束的中断请求线 IR 的编码。

6） PIC的初始化与操作命令

PIC 的命令共有 4 个，分为两类。具体如表 5-14 所示。

表 5-15 IA-32 中 8259A 命令与端口关系表

地址	操作	含　义	特征位 D4D3D2	时　序
50H	写	ICW1	1XX	无
		OCW2	00X	
52H	写	ICW2	无	在初始化期间，ICW1 之后 ICW2
		OCW1		非初始化期间
50H	读	IRR	无	无
52H	读	IMR	无	无

（1）　初始化命令（ICW1～ICW2）

a）初始化命令 ICW1

此命令进入初始化状态。

b）初始化命令 ICW2

此命令用于中断向量号的设置。PIC 提供给 CPU 的中断向量号（8 位），是在 CPU 对 PIC 初始化时，通过初始化命令 ICW2 设置的。初始化时，只需对高 5 位进行设置，而低 3 位就是 PIC 外部引脚 IR*i* 中的 *i*。例如，ICW2 设置为 08H，则 IR1 的中断向量号就是 08H+1=09H，而 IR7 的中断向量号就是 08H+7=0FH。在中断过程中，当第一个 INTA#到来时，自动将引脚 IR*i* 的 *i* 写入低 3 位。ICW2 的格式如图 5-81 所示。

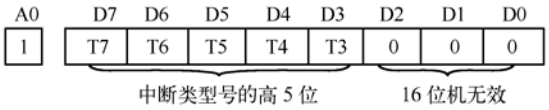


图 5-82 ICW2 格式

（2）　操作命令（OCW1～OCW2）

a）操作命令 OCW1

此命令用于中断屏蔽操作,PIC 使用的是常规屏蔽方式

为了对中断进行控制，通过 PIC 的中断屏蔽寄存器（IMR），可以屏蔽一个或几个中断请求。这样，可以在不改变硬件结构的情况下，使得主程序的不同部分可以使用不同的中断；还可以在中断服务子程序中，屏蔽一些中断（包括比自己优先级高的中断），实际上就是改变了中断的优先级。

常规屏蔽方式是常用的一种屏蔽方式，其屏蔽命令的格式如图 5-82 所示。

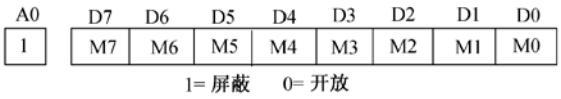


图 5-83 OCW1 格式

b）操作命令 OCW2

OCW2 的格式如图 5-83 所示。

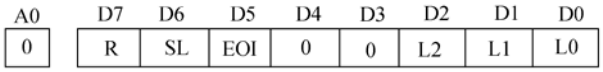


图 5-84 OCW2 格式

- D2～D0：L2～L0 这 3 位编码用来指定中断等级（0～7）。
- D3, D4 为 00，是特征位。
- D5（EOI）= 1，当中断服务程序执行完时，则需要给 PIC 一个通知，PIC 将此次中断在 ISR 寄存器所置的位清零。
- D6（SL）=1，指定非自动中断结束（EOI）方式，在此方式时，利用 OCW2 中的 L2～L0 的编码来指定某一个中断结束。
- D7（R）=0，不采用优先级轮换方式，即优先级固定；

OCW2 被用做中断结束操作，SL=1，L2～L0 编码是被指定的中断等级，OCW2= 01100L2L1L0。

7） PIC的中断响应周期

当 PIC 向 CPU 发中断后，CPU 会回应 PIC，这一过程称为 PIC 的中断响应周期，其详细过程如下：

① 一条或多条外部中断请求线（IR7～IR0）产生正跳变，使中断请求寄存器（IRR）相应位置“1”。

② PIC 接收这些中断请求，分析它们的优先级，向 CPU 发出请求信号 INT，并记录中断向量号。

③ 当 CPU 从 INTR 引脚收到 PIC 的中断请求信号 INT 后，如果当前的指令已经执行完毕，而且中断标志位 EFLAGS. IF＝1（中断允许），则 CPU 进入中断响应周期。

④ CPU 响应中断，并以 INTA#脉冲作为回答。

⑤ PIC 接收来自 CPU 的 INTA#脉冲，最高优先级的 ISR 位置“1”，而相应的 IRR 位被

复位，同时送出中断向量号。

8） 接口信号

本实验中相应接口信号定义如表 5-15 所示：

表 5-16 PIC 模块接口信号定义

信号名称	位宽	类型	意义
INTA_N	1	输入	中断响应信号
A	1	输入	地址信号
IR	8	输入	外设中断请求信号
DB_I	8	输入	数据信号（输入）
INT	1	输出	中断请求信号
DB_O	8	输出	数据信号（输出）
CS_N	1	输入	片选信号
IOR_N	1	输入	I/O 读信号
IOW_N	1	输入	I/O 写信号

各个信号具体功能描述如下：

1、INTA_N：中断响应信号。通知 PIC 中断请求已被响应

2、A：地址信号。用作低位地址配合片选信号来区分不同的端口。

3、IR：外设请求信号。

4、DB_I：数据信号（输入）。输入 PIC 的数据。

5、INT：中断请求信号。用以对 CPU 提出中断请求。

6、DB_O：数据信号（输出）。从 PIC 输出的数据。

7、CS_N：片选信号。

8、IOR_N：I/O 读信号。低电平有效。

9、IOW_N：I/O 写信号。低电平有效。

5. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface \Ex_6\Ex_6_PIC\Ex_6_PIC.xpr，打开工程。

2） 完成模块设计

点击工程目录中 PIC.v，根据文件中的提示，完成相关设计。

3） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

4） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

5） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

6） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑。注意此处的输入多路选择器需要加入 ITNA_N 信号用以判断何时将从 PIC 获取的中断向量送出，如图 5-84 所示。

Muxdati.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  module Muxdati(
4      input wb_tga_i,
5      input RDN,
6      input INTA_N,
7      input [15:0] MEMDATI,
8      input IOCS0_N,
9      input [15:0] IODATOI,
10     input IOCS1_N,
11     input [15:0] IODAT1I,
12     output [15:0] DATO
13     );
14
15     assign DATO = (!INTA_N) ? IODAT1I : //PIC中断向量号
```

图 5-85 输入多路选择器修改

7） **写 S86 汇编程序，完成实验要求中的功能**

8） 编译、链接、转换变成coe加入工程文件中

9） 仿真

添加仿真 coe，为了统一仿真标准，请先将 C:\sysclassfiles\interface \Ex_6\ sim.coe 加载在内存中，然后再进行仿真。

点击 Simulation 下的 Run Simulation，点击 Run Behavior Simulation 进行行为仿真，点击，开始仿真。

可以看到如图 5-85 和图 5-86 所示的仿真结果。IRi 上升沿产生 INT，INT 上升沿保存 IVN，INTA_N 下降沿清 IRR，置位 ISR，中断采用非自动结束。

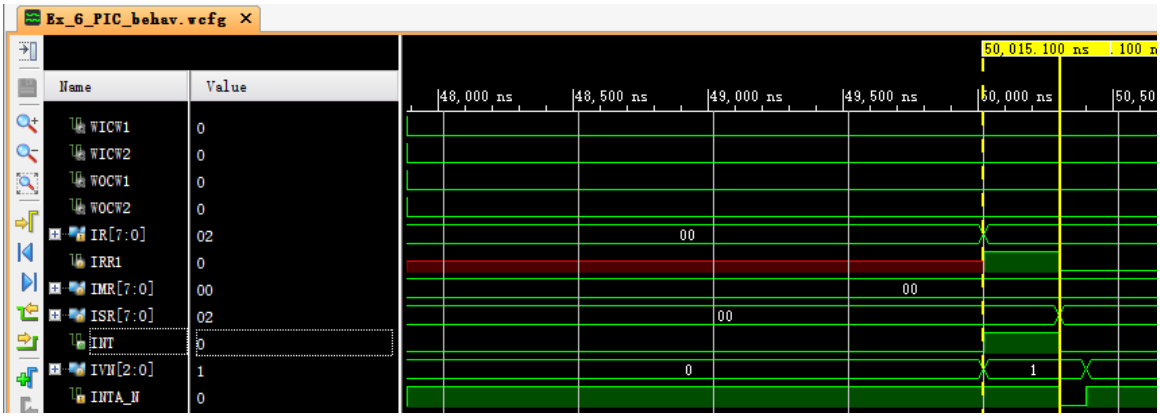


图 5-86 仿真结果图 1

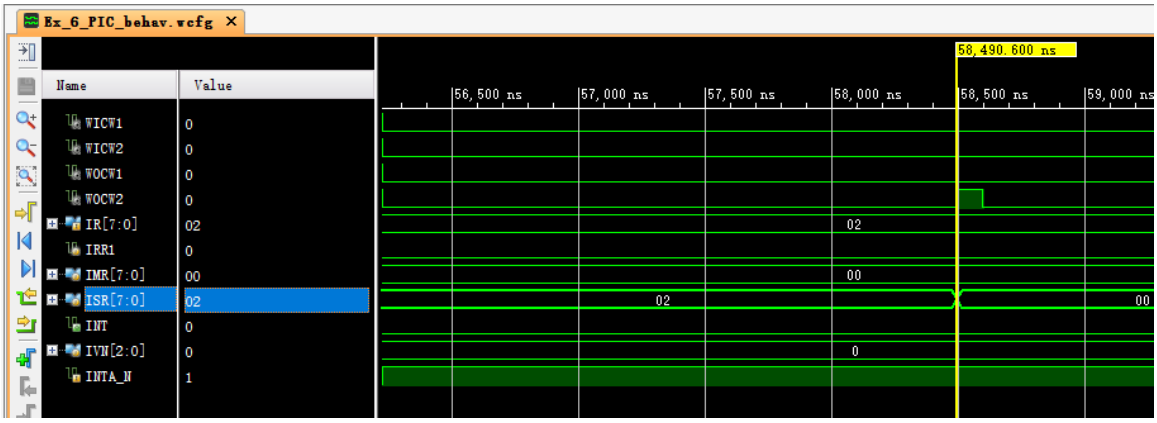


图 5-87 仿真结果图 2

10）　综合、实现、产生比特流文件、下载和运行

6. 实验现象

满足实验要求，能够体现优先级嵌套和屏蔽。

7. 思考与研讨

比较 PIC 与 8259 的异同点，试分析 8259 设计的合理性。

5.7　类 8255 可编程并行接口设计

1. 实验目的

设计一个类 8255 可编程并行接口并学会对其编程。

2. 实验内容及要求

1)　完成一个类 8255 可编程并行接口。要求具有以下功能：

（1）A 口方式 1 输入，B 口方式 0 输出，C 口只支持 A 口方式 1 输入的控制与应答信号。对 C 口支持按位置和按位清。

（2）满足 8255 方式 1 以及方式 0 时序，时间间隔不严格控制。

2)　编写汇编程序完成类 8255 可编程并行接口初始化，从 A 口读数据送至 B 口。

3. 实验预习

认真复习 8255 方式 0 和方式 1.

4. 实验原理

微机与 I/O 设备的接口按照数据传送方式的不同，可分为并行接口与串行接口两种。

并行接口最基本的特点是，一个待传送数据的各位同时传输，即在 CPU 与 I/O 设备之间传送信息的数据单位一般为“字节”或“字”，需要使用多根数据线，如打印机并行接口等。CPU 与 I/O 设备之间使用串行接口传送信息的数据单位为“位”，即一次传输一个信息位，只需要一根数据线即可进行串行传送，如 USB、键盘及调制解调器接口等。显然，并行和串行，仅指 I/O 接口与 I/O 设备之间的通信方式，而对于接口与 CPU 之间的通信来说，并行接口与串行接口一样，传送信息的数据单位均为字节或字。

并行接口的优点是传输速率高，但由于需要多根数据线，不适合长距离数据传输，一般用于近距离传送的场合。另外，并行传送的信息，一般不要求固定的格式，这与串行传送的信息有数据格式的要求不同。

并行接口电路有不可编程接口和可编程接口之分。可编程接口使用灵活，功能很强。8255 就是在微机系统中广泛应用的可编程并行接口。其基本功能如下。

① 8255 具有 2 个独立的 8 位 I/O 口（A 口和 B 口）和 2 个独立的 4 位 I/O 口（C 口上半部和 C 口下半部），提供与 TTL 兼容的并行接口。作为输入时提供三态缓冲器功能，作为输出时提供数据锁存功能。其中，A 口具有双向传输功能。

② 8255 有 3 种工作方式，即方式 0、方式 1 和方式 2；能使用多种数据传送方式完成 CPU 与 I/O 设备之间的数据交换，如无条件方式、查询方式和中断方式。

③ B 口和 C 口的引脚具有达林顿复合晶体管驱动能力，在 1.5 V 时输出 1 mA 电流，适于做输出端口。

④ C 口除用做数据口外，当 8255 工作在方式 1 和方式 2 时，C 口的部分引脚作为固定的联络信号线。

1）　内部结构

如图 5-87 所示，8255 内部结构由以下四部分构成。

（1）　数据总线缓冲器

数据总线缓冲器是双向三态 8 位缓冲器，可以直接与系统数据总线相连，实现 CPU 和端口之间的信息交换。所有数据的发送和接收，以及 CPU 发出的命令字和从 8255 来的状态信息都是通过该缓冲器传送的。

（2）　读/写控制模块

地址线 Al 和 A0，片选信号 CS#和读/写控制信号（RD#，WR#），完成内部端口的选择和读/写操作。

（3）　A组和B组控制模块

控制电路内部设有控制寄存器，可以根据 CPU 送来的控制命令来控制 8255 的工作方式，也可以根据编程命令来对 C 口的每一位进行置/复位的操作。A 组控制模块管理 A 口和 C 口的高 4 位（PC7～PC4），B 组控制模块管理 B 口和 C 口的低 4 位（PC3～PC0）。

（4）　I/0端口

I/O 通道由 3 个 8 位的端口寄存器，即 A 口、B 口和 C 口组成，它们都可编程为输入/输出，而且都有数据锁存功能。C 口可通过编程分为两个 4 位口，每一个 4 位口都可定义为输入口或输出口，用于传送数据。

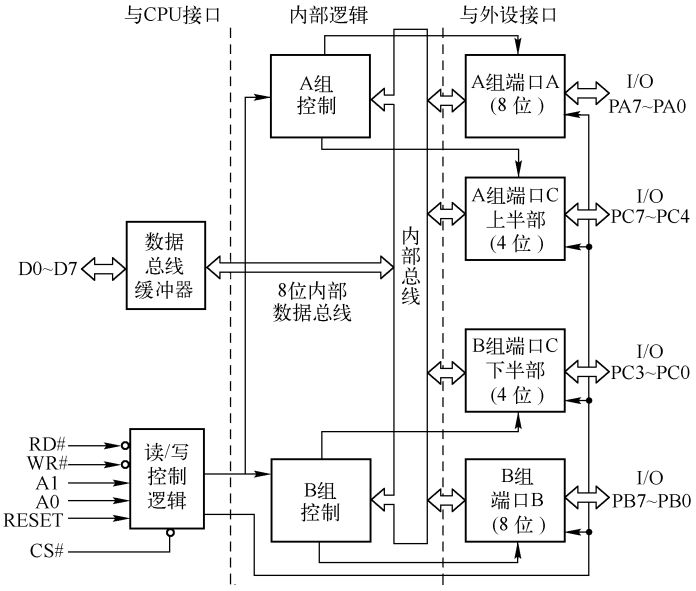


图 5-88 8255 内部结构框图

2） 外部引脚

8255 是一个单+5 V 电源供电、40 个引脚的双列直插式组件，其外部引线如图 5-88 所示。

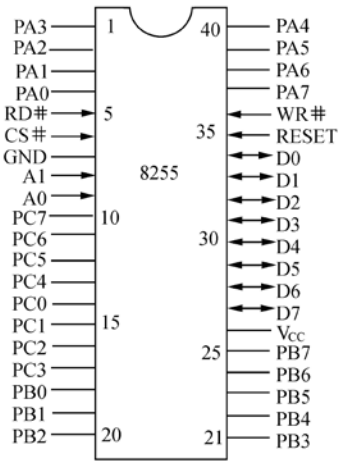


图 5-89 8255 引脚图

作为接口电路的 8255 具有面向 CPU 和面向外设两个方向的连接能力。因此，它的引脚分成以下两部分。

（1） 与系统总线的连接信号

D7～D0——双向数据线，用于 CPU 向 8255 发送命令和数据，以及 8255 向 CPU 回送状态和数据。

CS#——选片信号，低电平有效，由系统的高位地址线经 I/O 端口地址译码电路产生。

A1 和 A0——芯片内部端口地址信号，与系统地址总线低位相连，用来寻址 8255 内部寄存器。两位地址可形成片内 4 个端口地址。

RD#和 WR#——读/写信号，低电平有效。CPU 通过执行 IN 指令使 RD#有效，即发读信号将数据或状态信息从 8255 读至 CPU；CPU 通过执行 OUT 指令使 WR#有效，即发写信号，将命令字或数据写入 8255。

RESET——复位信号，高电平有效。它清除控制寄存器并将 8255 的 A, B 和 C 三个端口均置为输入方式；输出寄存器和状态寄存器被复位，并且屏蔽中断请求；24 条面向外设的信号线呈现高阻悬浮状态，这种状态一直维持到工作方式控制字被设置，使得 8255 进入用户所需的工作方式。

IA-32 中 8255 的端口地址为 60H~63H，各端口与操作选择表如表 5-16 所示。

表 5-17 IA-32 中 8255 的端口与操作选择表

A1	A0	RD#	WR#	CS#	I/O 地址	操　作	特征位	方向
0	0	0	1	0	60H	A 口内容读至数据总线	无	输入
0	1	0	1	0	61H	B 口内容读至数据总线		
1	0	0	1	0	62H	C 口内容读至数据总线		
0	0	1	0	0	60H	数据总线内容写至 A 口	无	输出
0	1	1	0	0	61H	数据总线内容写至 B 口		
1	0	1	0	0	62H	数据总线内容写至 C 口		
1	1	1	0	0	63H	数据总线内容写至控制寄存器	D7=1	
						数据总线内容写至 C 口，按置位/复位控制字	D7=0	
X	X	X	X	1	—	端口输出为高阻	无	禁止
1	1	0	1	0	—	非法		
X	X	1	1	0	—	端口输出为高阻		

（2）　与外部设备的连接信号

PA7～PA0 为端口 A 的输入/输出线，PB7～PB0 为端口 B 的输入/输出线，PC7～PC0 为端口 C 的输入/输出线。这 24 根信号线均可用来连接 I/O 设备和传送信息。其中，A 口和 B 口通常只作为输入/输出的数据口，尽管有时也利用它们从 I/O 设备读取一些状态信号，这些信号也是作为数据读入的。A 口和 B 口作为数据口输入/输出时，是 8 位一起行动的，即使只用到其中的某 l 位，也要同时输入/输出 8 位。

C 口的作用与 8255 的工作方式有关，它除了作为数据口以外，还有其他用途，故 C 口的使用比较特殊，单独介绍如下：

① 作为数据口。C 口作为数据口时和 A 口、B 口不一样，它是把 8 位分成高 4 位和低 4 位两部分。高 4 位 PC7～PC4 与 A 口一起组成 A 组，低 4 位 PC3～PC0 与 B 口组成 B 组。因此，C 口作为数据口输入/输出时，是 4 位一起行动，即使只使用其中的 l 位，也要 4 位一起输入或输出。

② 作为状态口。8255 在方式 1 和方式 2 下，有固定的状态字，是从 C 口读入的。此时，C 口就是 8255 的状态口。而 A 口和 B 口不能作为 8255 本身的状态口。

③ 作为专用握手信号线。8255 的方式 1 和方式 2 是应答方式，在传送过程中需要进行应答的握手信号，而 C 口的大部分引脚此时就被分配作为固定的握手线。

④ 做按位控制用。C 口的 8 个引脚可以单独从 l 个引脚输出高/低电平。此时，C 口用做按位控制，而不是用做数据输出。

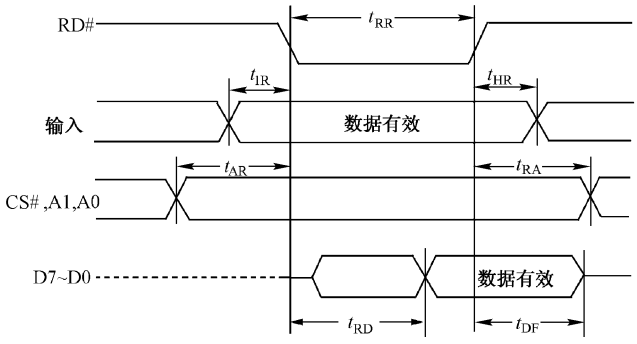
3） 8255 的工作方式

8255 一共有三种工作方式。

（1）　方式 0

方式 0 为一种简单的输入/输出方式，在这种方式下，A, B, C 三个口中的任何一个都可进行输入或输出操作，它不需要应答式握手信号，外设总是处于准备好的状态。方式 0 提供两个 8 位口（A 和 B）和两个 4 位口（PC7～PC4，PC3～PC0），由 CPU 用简单的 I/O 指令来进行读/写。方式 0 中各口都是单向传送的，一次初始化只能设置在一个方向上传送数据。方式 0 一般用于无条件传送的场合，也可以用做查询式传送。当选用 8255 的方式 0 作为查询式接口电路时，通常将 A 口和 B 口作为数据口，把 C 口的 4 位（高 4 位或低 4 位）规定为输出口，用以输出一些控制信号，把 C 口的另 4 位规定为输入口，用以读入外设的状态。

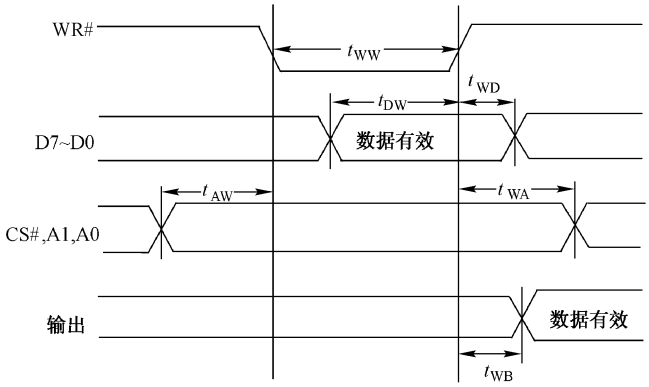
8255 工作于方式 0 作为输入时其时序见图 5-89，作为输出时其时序见图 5-90（两图均以 8255A 为例）。



符号	参　数	8255A　（单位：ns）	
		最小值	最大值
t_{RR}	读脉冲宽度	300	

t_{IR}	输入领先于 RD#的时间	0	
t_{HR}	输入滞后于 RD#的时间	0	
t_{AR}	地址稳定领先读信号的时间	0	
t_{RA}	读信号无效后地址保持时间	0	
t_{RD}	从读信号有效到数据稳定		250
t_{DF}	读信号去除后至数据浮空	10	150
t_{RY}	在两次读（或写）之间的时间间隔	850	

图 5-90 8255A 工作方式 0 的输入时序和参数说明



符号	参　数	8255A （单位：ns）	
		最小值	最大值
t_{AW}	地址稳定领先写信号的时间	0	
t_{WA}	写信号后地址保持时间	20	
t_{WW}	写脉冲宽度	400	
t_{DW}	从写信号到数据有效	100	
t_{WD}	数据保持时间	30	
t_{WB}	从写信号结束到输出		350

图 5-91 8255A 工作方式 0 的输出时序和参数说明

（2）　方式 1

方式 1 是一种选通输入/输出方式。A 口和 B 口均可工作在这种方式。在这种方式下，A 口和 B 口仍作为两个独立的 8 位 I/O 数据通道（可连接外设），C 口要有 6 位（分成两个 3 位）分别作为 A 口和 B 口的应答握手线，其余 2 位仍可作为方式 0 的输入/输出。方式 1 也是单向传送的。

a）方式 l 的输入

图 5-91 给出了 8255 的 A 口和 B 口在方式 l 的输入组态。

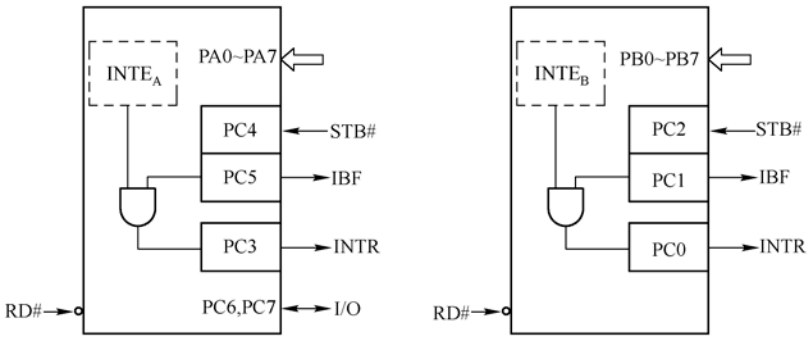


图 5-92 8255 方式 1 输入组态

从组态中我们可以看出，C 口的 PC3～PC5 用做 A 口的应答握手线，而 PC0～PC2 则用做 B 口的应答握手线，余下的 PC6 和 PC7 则可作为方式 0 使用。应答握手线的功能如下：

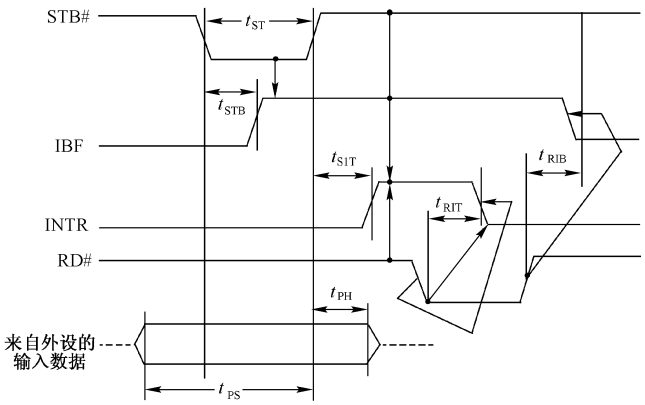
STB#——选通输入信号，低电平有效。当其有效时，把输入装置来的数据送入 8255 输入缓冲器。

IBF——输入缓冲器满信号，高电平有效。这是 8255 输出给外设的握手信号，作为 STB# 的应答信号。当其有效时，表示数据已装进输入缓冲器，但 CPU 还未将数据取走，通知外设停止送数。当 CPU 将数取走以后，IBF 复位，外设可继续送数。它由 STB#信号置位，由 RD#信号的后沿（上升沿）使其复位。

INTR——中断请求信号，高电平有效。它是 8255 向 CPU 发出的中断请求信号，当 STB#, IBF 和 INTE（中断允许）均为高电平时被置高，由 RD#信号的下降沿清除。

INTE——中断允许信号，高电平有效。只有在 INTE=l 时，A 口和 B 口才能因输入缓冲器满向 CPU 发中断申请信号。用户通过 PC4 的置位控制字来使 $INTE_A$ 置 l，通过 PC2 的置位控制字使 $INTE_B$ 置 l，使 A 口和 B 口允许中断。使用 PC4 或 PC2 的复位控制字可使 $INTE_A$ 或 $INTE_B$ 复位，以禁止中断。

当外设准备好数据，即数据已输至 8255 的端口数据线上时，就发出 STB#选通信号，将数据通过 A 口或 B 口锁存到 8255 的数据输入寄存器。选通信号变低后，8255 输出 IBF 表示输入缓冲器已满，阻止外设输入新的数据，并提供 CPU 查询；在选通信号结束后，8255 向 CPU 发出 INTR 中断请求信号（如果中断允许的话），CPU 响应中断，发出 RD#信号，把数据读入 CPU。RD#有效信号清除中断请求；然后 RD#信号结束，使 IBF 变低，表示输入缓冲器已空，通知外设可以输入新的数据。图 5-92 表示了上述过程并包含了每个状态时间范围。



符号	参　　数	8255A　（单位：ns）	
		最小值	最大值
t_{ST}	STB#脉冲宽度	500	
t_{STB}	STB#=0 到 IBF=1		300
t_{SIT}	STB#=1 到 INTR=1		300
t_{RIB}	RD#=1 到 IBF=0		300
t_{RIT}	RD#=0 到期 INTR=0		400
t_{PS}	数据提前 STB#无效的时间	0	
t_{PH}	数据保持时间	180	

图 5-93 8255 方式 1 输入时序和参数说明

采用查询式输入时，CPU 先查询 8255 的输入缓冲器是否满，即 IBF 是否为高。若 IBF 为高，则 CPU 就可以从 8255 读入数据。采用中断方式传送时，应该先用 C 口置位控制字，使相应端口允许中断，即使 PC4 或 PC2 置“1”。

b）方式 1 的输出

8255 的 A 口和 B 口工作于选通输出方式时，PC 口各线的作用如图 5-93 所示。

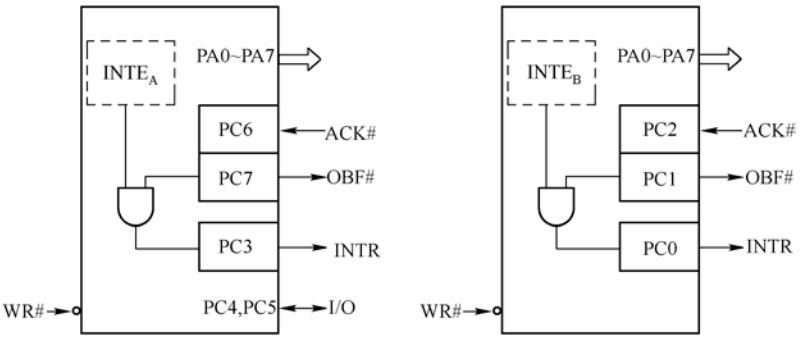


图 5-94 8255 方式 1 输出组态

PC6, PC7 和 PC3 分配给 A 口作为应答握手线，PC0～PC2 仍然作为 B 口的应答握手线，剩下的 PC4 和 PC5 可作为方式 0 使用。应答握手线的功能如下：

OBF#——输出缓冲器满信号，低电平有效。这是 8255 给外设的握手信号。当其有效时，表明 CPU 已经把数据输出给指定的端口，外设可以从 8255 取数。它由输出指令产生的 WR# 的上升沿置成有效，由 ACK#信号使其恢复为高。

ACK#——外设响应信号，低电平有效。作为对 OBF#的响应信号，表示外设已将数据从 8255 的输出缓冲器中取走。

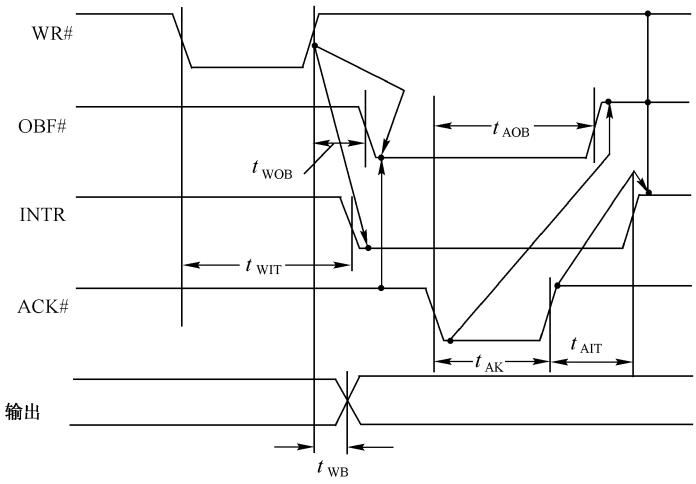
INTR——中断请求信号，高电平有效。当外设已经接受了 CPU 输出的数据后，8255 向 CPU 发出中断请求，要求 CPU 输出新的数据。当 INTE 为“1”并且 ACK#由低变高时 INTR 置位，而 WR#信号的上升沿使 INTR 复位。

INTE——中断允许信号，高电平有效。分别由 PC6 和 PC2 的置位/复位控制。

方式 l 输出时序如图 5-94 所示。CPU 输出数据，发出 WR#信号。WR#信号的下降沿将微处理器数据送到输出数据锁存器。WR#的上升沿有三个作用：一是将数据输出到 8255 的端口线上；二是使 OBF#信号有效，表明输出缓冲区已满，通知外设来取数据；三是清除中断请求信号。外设接收数据后发出 ACK#信号，它一方面使 OBF#无效，另一方面 ACK#上升沿使 INTR 有效，发出新的中断请求信号，让 CPU 输出新的数据。

使用查询方式输出时，CPU 在输出数据后查询 OBF#是否变高。若变高则表明输出缓冲器空，即数据已被外设接收，可以输出新的数据。若使用中断方式传送，要使中断得以发生，

则必须先使用 C 口置位控制字使之允许中断，即使 PC6 或 PC2 置“1”。



符号	参　　数	8255A　（单位：ns）	
		最小值	最大值
t_{WOB}	WR#=1 到 OBF#=0		650
t_{WIT}	WR#=0 到 INTR=0		850
t_{AOB}	ACK#=0 到 OBF#=1		350
t_{AK}	ACK#脉冲宽度	300	
t_{AIT}	ACK#=1 到 INTR=1		350
t_{WB}	WR#=1 到输出		350

图 5-95 8255 方式 1 输出时序和参数说明

（3）　方式 2

方式 2 为双向选通 I/O 方式，只有 A 口才有此方式。这时，C 口有 5 根线用做 A 口的应答握手信号，其余 3 根线可用做方式 0，也可用做 B 口方式 l 的应答握手线。图 5-95 给出了方式 2 的组态。

方式 2 实际上就是 A 口方式 1 的输入与输出方式的组合，各应答信号的功能也相同。输入/输出的先后顺序是任意的，根据实际传送数据的需要选定。输出过程由 CPU 执行输出指令向 8255 写（WR#）数据开始，然后，外设从 8255 取数，并返回 ACK#。而输入过程则是从外设向 8255 发选通信号 STB#开始，然后，CPU 执行输入指令，从 8255 读（RD#）数据。方式 2 本实验不涉及。

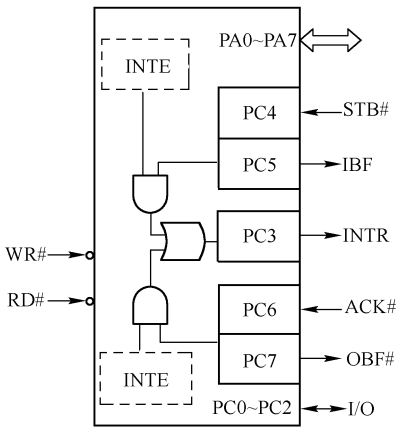


图 5-96 8255 方式 2 的组态

4） 8255 的控制字与初始化编程

8255 的工作方式和接口功能是由 CPU 通过把控制字写入控制寄存器来决定的，这就是初始化编程。8255 的控制字有两个，一个是工作方式控制字，另一个是 C 口按位置/复位控制字。这两个控制字都要写入控制寄存器，通过标志位（D7）来区分。

（1） 工作方式控制字

工作方式控制字的格式如图 5-96 所示（注意最高位一定为 1）。由图可知：

① A 组有三种工作方式（方式 0、方式 1 和方式 2），B 组只有两种工作方式（方式 0 和方式 1），C 口除用于固定的应答握手线外的所有信号线均工作在方式 0。

② 端口 A 和端口 B 要分别作为一个整体确定工作方式，而端口 C 则是分成高 4 位和低 4 位两部分分别确定工作方式。端口 A 和端口 B 的工作方式可以不同，端口 C 的上半部和下半部的工作方式也可以不同。8255 这四部分的工作方式可以任意组合。

D7	D6	D5	D4	D3	D2	D1	D0
标志位=1	A 组方式选择 00：方式 0 01：方式 1 1X：方式 2		A 口 0：输出 1：输入	C 口（高 4 位） 0：输出 1：输入	B 组方式选择 0：方式 0 1：方式 1	B 口 0：输出 1：输入	C 口（低 4 位） 0：输出 1：输入

图 5-97 8255 工作方式控制字

（2） C口按位置/复位控制字

C 口按位置/复位控制字也是写入控制寄存器的一个控制字，而不是写入 C 口。其格式如图 5-97 所示（注意最高位一定为 0）。

C 口按位置/复位控制的功能有两个：一是用于对外设的控制，利用这一功能，可使 C 口某一位输出一个开关量或一个脉冲，作为外设的启动或停止信号。二是可用于设置方式 1 和方式 2 的中断允许。此时，C 口按位置/复位控制字不影响对应的引脚状态，只是起到设置 INTE、开关 8255 中断的作用。

D7	D6	D5	D4	D3	D2	D1	D0
标志位=0	不用（一般置 0）			C 口的位选择 000：C 口位 0 001：C 口位 1 ⋮ 111：C 口位 7			1=置位 0=复位

图 5-98 C 口按位置/复位控制字

综上所述，8255 的初始化编程比较简单。一般方式 0 可作为无条件传送方式，也可作为查询方式使用，此时 8255 与外设的握手信号最灵活。方式 1 可作为查询方式使用，此时握手信号线固定；方式 1 也可用做中断方式，这时，注意一定要写对应的 C 口按位置位字，打开中断。

5. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface \Ex_7\EX_7_S8255 \ EX_7_S8255.xpr，打开工程。

2） 完成模块设计

点击工程目录中 S_8255.v，根据文件中的提示，完成相关设计。

3） 将 3.6 设计模块或者将课程资源IP核加入工程

课程资源 IP 核为 C:\sysclassfiles\interface\IP_Peripheral 中的 CSE_PIC_1.0 和 CSE_S8255test_1.0。

CSE_S8255test_1.0 是用来测试方式 1 时序的，当时序有问题时，对应的错误指示灯将会亮。

4） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

5） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

6） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

7） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑。

8） 写S86 汇编程序，完成实验要求中的功能

9） 编译、链接、转换变成coe加入工程文件中

10） 综合、实现、产生比特流文件、下载和运行

6. 实验现象

满足实验要求。指示灯（YLED7~YLED5）全灭，且 ibfa 熄灭时将对应的 SW7~SW0 输出到 GLED7~GLED0。

7. 思考与研讨

完成 A 口方式 1 输出，A 口和 B 口方式 0 输入/输出模式。

5.8　交通灯控制实验

5.8.1　红黄绿三色交通灯控制

1. 实验目的

通过类 8255 并行接口、类 8254 定时/计数器以及可编程中断控制器 PIC 实现十字路口交通灯的模拟控制，进一步掌握对定时计数器、并行口的使用和中断控制器的综合使用。

2. 实验内容

1) 十字路口交通灯的变化规律

- 南北路口的绿灯、东西路口的红灯同时亮 10 秒，然后
- 南北路口的黄灯闪烁 2 次（闪烁周期 2 秒），同时东西路口的红灯继续亮，然后
- 南北路口的红灯、东西路口的绿灯同时亮 10 秒，然后
- 南北路口的红灯继续亮，同时东西路口的黄灯 2 次（闪烁周期 2 秒），然后
- 转 1 重复。

2) 2、将实验板子 RD1、YD1、GD1 作为南北路口的交通灯与类 8255 的 PA7、PB6、PA5 相连；RD0、YD0、GD0 作为东西路口的交通灯与类 8255 的 PA2、PA1、PA0 相连(方式 0)。利用 8254 产生 1 秒的中断信号，在中断处理程序中用程序处理十字路口交通灯的变化的问题。

3. 实验预习

认真考虑交通灯变化规律，画出流程图和状态转换图。

4. 实验原理

对于 8254 中断，由于是 1 秒一次中断，所以中断处理程序需要有四个状态：S1、S2、S3、S4。S1 状态是南北方向延迟 10 秒的正常周期状态，需要有变量来计数 10 次，以便延迟 10 秒；S2 是南北方向黄灯闪烁周期，利用一个变量来计数形成 4 秒的闪烁（闪烁周期 2 秒，闪烁 2 次）。这两个状态东西方向都是红灯。S3 状态是东西方向延迟 10 秒的正常周期状态，利用变量来计数 10 次，以便延迟 10 秒；S4 是东西方向黄灯闪烁周期，利用一个变量来计数形成 4 秒的闪烁（闪烁周期 2 秒，闪烁 2 次）。这两个状态南北方向都是红灯。其状态变迁图如图 5-98 所示。在 S2 状态和 S4 状态，需要解决黄灯闪烁问题，这个可以用另一个状态标志来表示当前是黄灯亮还是黄灯灭。

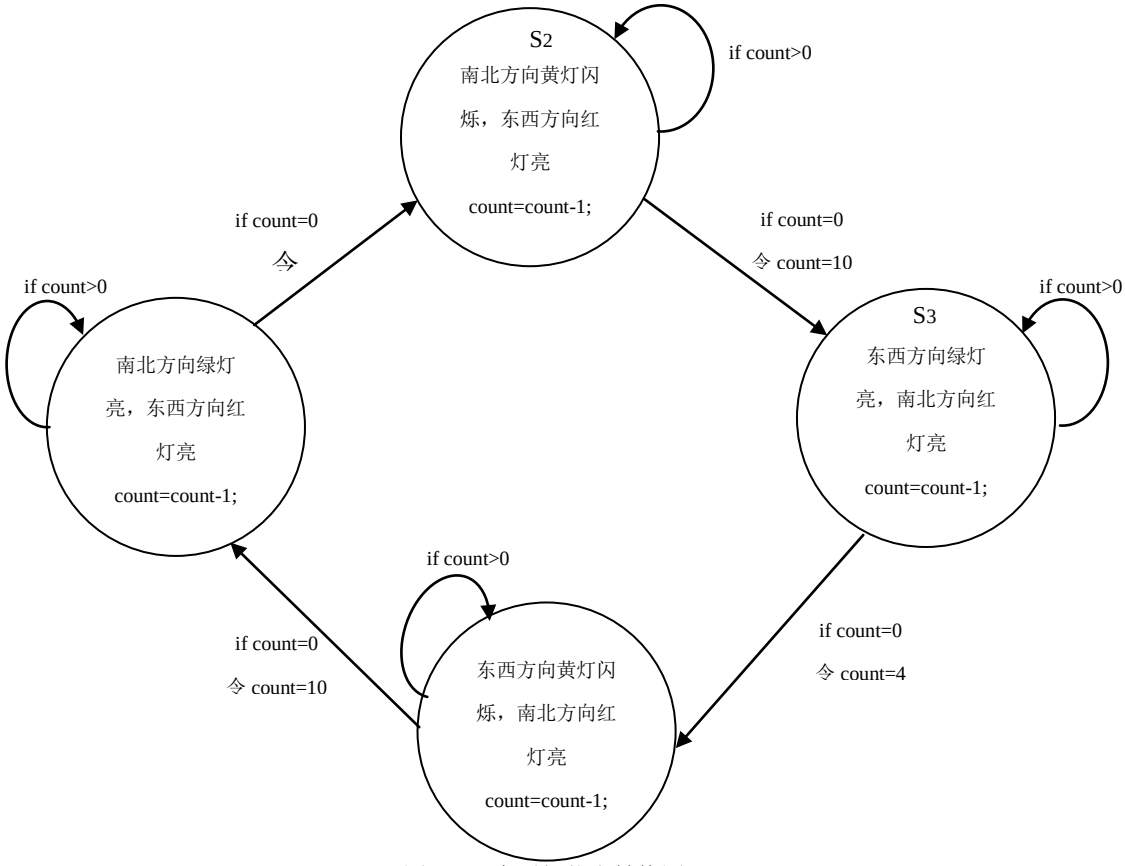


图 5-99 交通灯状态转换图

5. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface \Ex_8\Ex_8_trafficlight0\ Ex_8_trafficlight0.xpr，打开工程。

2） 将 5.5、5.6、5.7 设计模块或者将课程资源IP核加入工程

课程资源 IP 核为 C:\sysclassfiles\interface\IP_Peripheral 中的 CSE_PIC_1.0 、CSE_S_8254_1.0 和 CSE_S_8255_1.0。

3） 完成LED到 8255 的连接

4） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

5） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

6） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

7） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑。

8） 编写S86 汇编程序，完成实验要求中的功能

9） 编译、链接、转换变成coe加入工程文件中

10） 综合、实现、产生比特流文件、下载和运行

6. 实验现象

满足实验要求。

5.8.2 带倒计时的交通灯控制。

1. 实验目的

通过类 8255 并行接口、类 8254 定时/计数器以及可编程中断控制器 PIC 实现带倒计时十字路口交通灯的模拟控制，进一步掌握对定时计数器、并行口的使用和中断控制器的综合使用。

2. 实验内容

1) 十字路口交通灯的变化规律
   - 南北路口的绿灯、东西路口的红灯同时亮 10 秒，然后
   - 南北路口的黄灯闪烁 2 次（闪烁周期 2 秒），同时东西路口的红灯继续亮，然后
   - 南北路口的红灯、东西路口的绿灯同时亮 10 秒，然后
   - 南北路口的红灯继续亮，同时东西路口的黄灯 2 次（闪烁周期 2 秒），然后
   - 转 1 重复。
2) 将实验板子 RD1、YD1、GD1 作为南北路口的交通灯与类 8255 的 PA7、PB6、PA5 相连；RD0、YD0、GD0 作为东西路口的交通灯与类 8255 的 PA2、PA1、PA0 相连(方式 0)。PC 口的 PB0~PB6 作为输出口(方式 0 输出)连接 7 段数码管的段码，PC0、PC1 连接数码管的 A0，A1 来选择显示的位。利用 8254 产生 1 秒的中断信号，在中断处理程序中用程序处理十字路口交通灯变换的问题。同时数码管显示倒计时的值（10～0，4～0）。

3. 实验原理

同实验一。

4. 实验步骤

1） 打开工程

双击 C:\sysclassfiles\interface \Ex_8\Ex_8_trafficlight0\ Ex_8_trafficlight0.xpr，打开工程。

2） 将 3.5、3.6、3.7 设计模块或者将课程资源IP核加入工程

课程资源 IP 核为 C:\sysclassfiles\interface\IP_Peripheral 中的 CSE_PIC_1.0 、CSE_S_8254_1.0 和 CSE_S_8255_1.0。

3） 完成数码管、SW、LED到 8255 的连接

4） 完成地址译码器端口连接

根据表 2-5 中规定的外设的地址范围，连接 I/O 片选信号、地址信号和 I/O 读写信号。

5） 完成模块连接

连接系统总线上与模块相关信号（读写信号、数据线）。

6） 应答信号发生器修改

在应答信号发生器上将片选信号和相应逻辑加上。

7） 输入多路选择器修改

在输入数据多路选择器上加入新的端口和逻辑。

8） 写S86 汇编程序，完成实验要求中的功能

9） 编译、链接、转换变成coe加入工程文件中

10） 综合、实现、产生比特流文件、下载和运行

5. 实验现象

满足实验要求。

6. 思考与研讨

1) 请将 3.5、3.6、3.7 中自己设计的模块进行扩展后替换掉本实验中由资源包提供的 IP 核。

2) 实现可变化时长的带倒计时的交通灯控制。要求如下：

（1）将实验板子 RD1、YD1、GD1 作为南北路口的交通灯；RD0、YD0、GD0 作为东西路口的交通灯。数码管 A0，A1 作为倒计时显示，数码管 A4、A5 作为设置时长时候的显示。SW0 为高，设置南北向绿灯时长 M；SW1 为高，设置东西向绿灯时长 N。通过 Verilog 的行为描述方法，设计 FPGA 控制电路完成对交通灯变换的控制，同时数码管显示倒计时的值（10～0，4～0）。设计软件和相应的软件接口电路，共同完成对 M 和 N 的设置。

（2）对 M 和 N 的设置。

a）M 和 N 初始值为 10；

b）将 SW0 置为 1，通过 4X4 键盘设置 M 值（99~10），此时可暂停交通等变换控制，SW0 从 1 到 0 完成设置。并继续暂停的交通灯控制。

c）将 SW1 置为 1，通过 4X4 键盘设置 N 值（99~10），此时可暂停交通等变换控制，SW1 从 1 到 0 完成设置。并继续暂停的交通灯控制。