



第3章 S800的嵌入式实验

- **3.1 实验一 时钟选择与 GPIO 实验**
(ref. B-chapter5, 10, C-chapter26,14)
- **3.2 实验二 I2C扩展及SYSTICK中断实验**
(ref. B-chapter3,18, C-chapter16,17,28)
- **3.3 实验三 UART串行通讯口实验**
(ref. B-chapter16, C-chapter30)

本章节参考资料:

- A. 自编讲义《嵌入式系统实验教程》
- B. Tiva™ TM4C1294NCPDT
Microcontroller Data Sheet
- C. TivaWare™ Peripheral Driver
Library User's Guide
- D. S800板介绍V0.65



实验二 I2C扩展及SYSTICK中断实验

■ 实验目的

- 了解I2C 总线标准及在TM4C1294 芯片的调用方法
- 掌握用I2C总线扩展GPIO芯片PCA9557及TCA6424 的方法
- 能够通过扩展GPIO 来输出点亮LED 及动态数码管
- 进一步熟悉SYSTICK 定时中断的使用，掌握利用软定时器模拟多任务切换的方法



实验二 I2C扩展及SYSTICK中断实验

■ 预备知识

- 3.2.1 I²C串行总线 (ref. B-chapter18, C-chapter16)
- 3.2.2 I²C转8位I/O扩展芯片PCA9557PWR (ref. pca9557.pdf)
- 3.2.3 I²C转24位I/O扩展芯片TCA6424 (ref. tca6424.pdf)



3.2.1 IIC串行总线（Inter-Integrated Circuit）

■ 概述

- IIC（或I²C、I2C）总线接口是Philips推出的一种简单的双向两线制串行总线方式，用于IC器件之间的通信
- I²C总线仅使用两个信号：双向串行数据线SDA和串行时钟线SCL。数据线上的信号与时钟同步
- 多个符合I2C总线标准的器件都可以通过同一条I2C总线进行通信，器件地址采用硬件设置方法，扩展简单灵活
- 通过软件寻址识别每个器件，不需要额外的地址译码器



IIC串行总线的基本概念

■ I2C总线的设备分类

- 主机 (Master) : 初始化发送、产生时钟信号和终止发送的器件, 如微控制器
- 从机 (Slave) : 被主机寻址的器件
- ※ 发送器: 本次传送中发送数据 (不包括地址和命令) 到总线的器件
- ※ 接收器: 本次传送中从总线接收数据 (不包括地址和命令) 的器件

■ I2C总线的操作模式

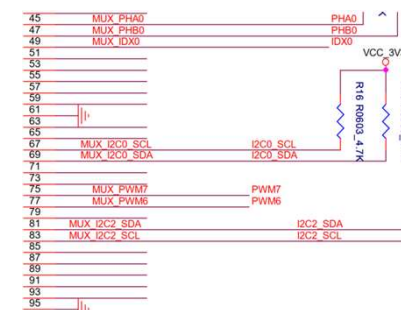
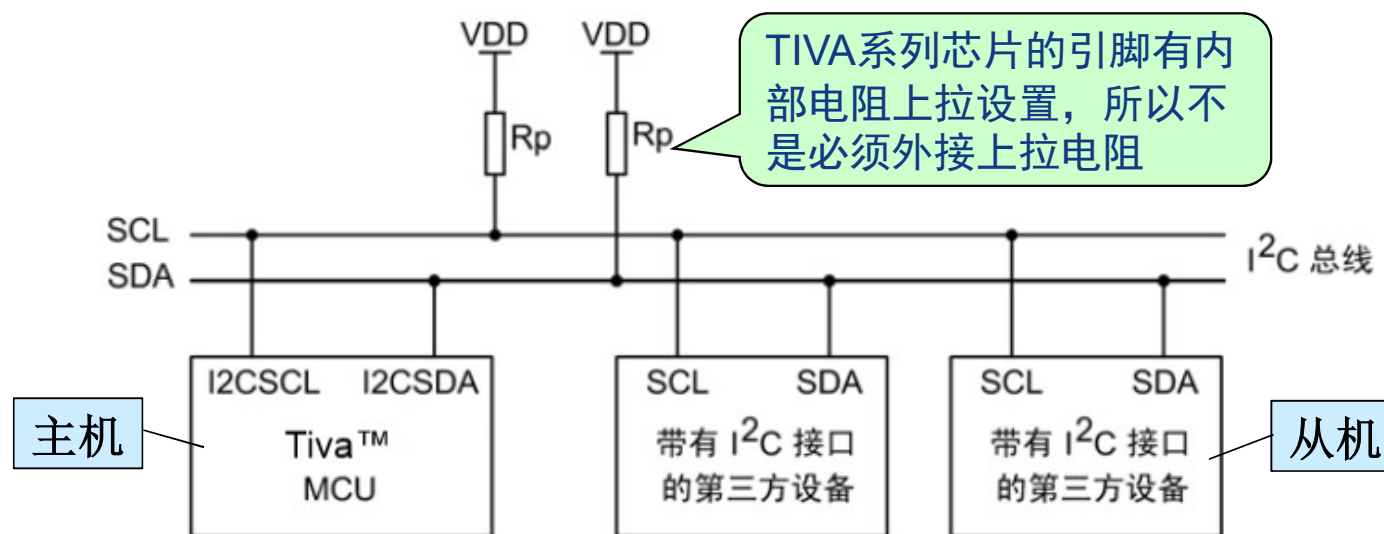
- 主发送、主接收、从发送、从接收

■ I2C总线接口的传输速率

- 标准100Kbit/s、快速400Kbit/s、快速附加1Mbit/s、高速3.33Mbit/s

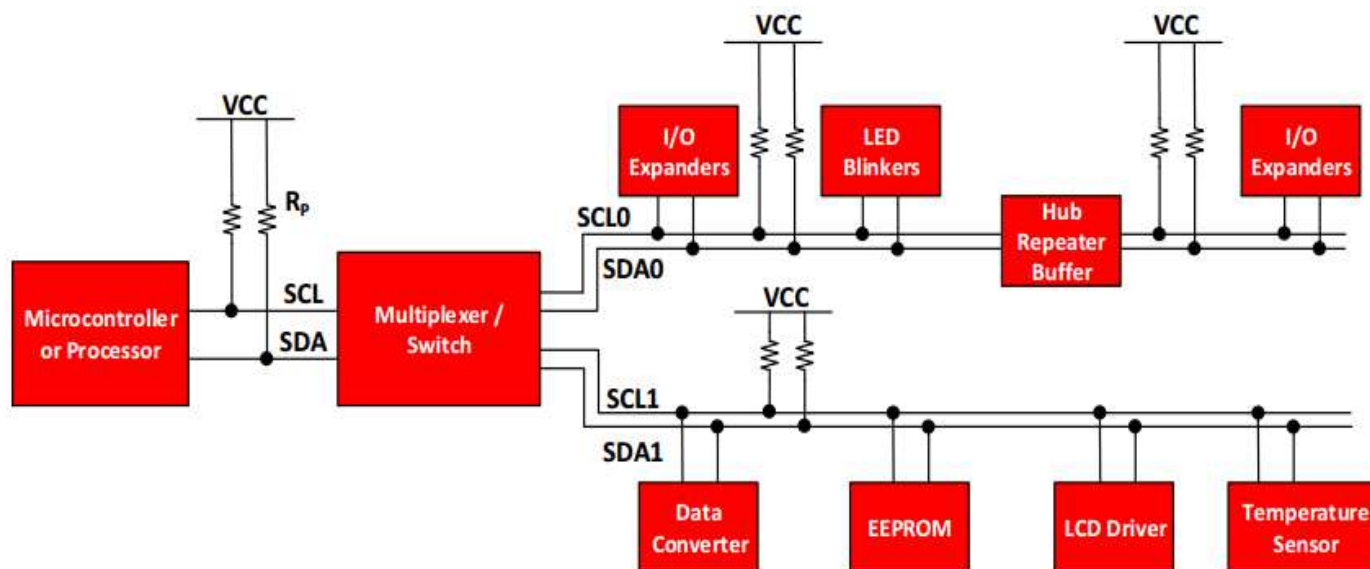
I2C信号线与连接方式

- I2C总线仅使用两个信号：SDA和SCL。SDA是双向串行数据线，SCL是串行时钟线。
- I2C总线接口均为开漏或开集电极输出，因此需要为总线增加上拉电阻 R_p ，且主机和从机之间要共GND。



I2C连接方式

- 器件地址采用硬件设置方法，使用 7 比特地址空间，有 16 个保留地址。因此理论上同一 I2C 总线能够支持的最大通信结点数是 $2^7 - 16 = 112$ 个





■ I²C总线会话过程

- 主机首先产生一个起始 (START) 条件;
- 主机发送它想访问的器件地址;
- 总线上的所有器件将接收到的地址与自己的地址相比较, 地址匹配成功的器件产生一个应答信号 (ACK)
- 主机收到应答信号后, 开始发送或接收数据
 - 数据传输格式: 最高有效位(MSB)在前、最低有效位(LSB)在后
- 当主机发送或接收完全部数据后, 产生一个停止 (STOP) 条件, 告知所有连接在总线上的设备总线被释放。



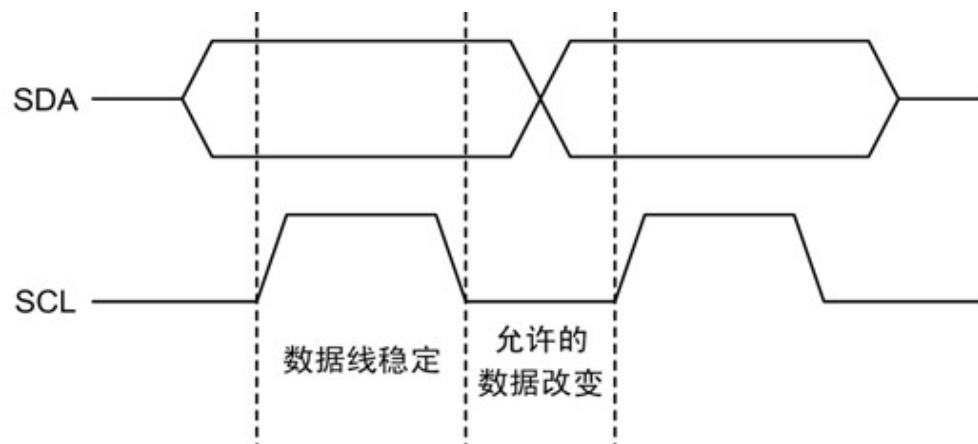
■ I2C总线时序

■ 总线空闲状态

- 当SDA和SCL线皆为高电平时，总线为空闲状态

■ 数据有效性

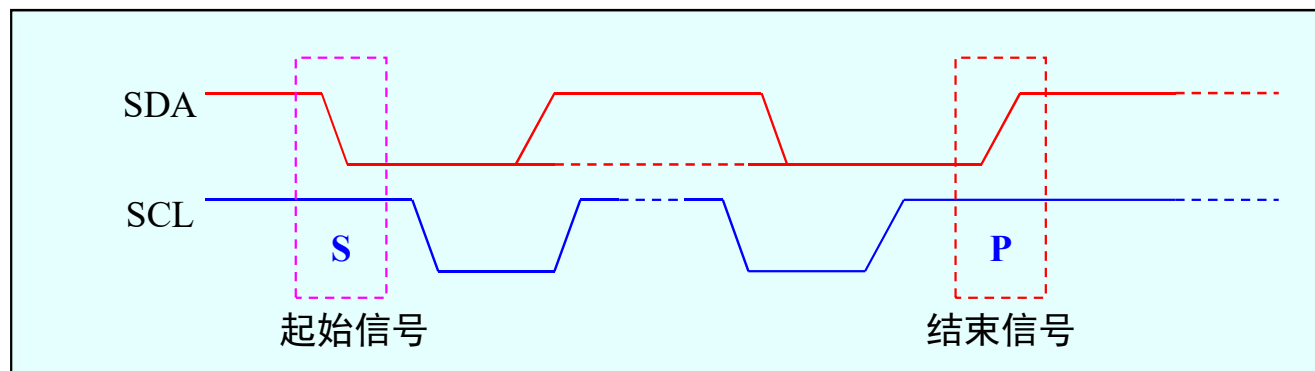
- 在时钟SCL的高电平期间，SDA线上的数据必须保持稳定。SDA仅可在时钟SCL为低电平时改变



■ 起始和停止条件

- 起始条件(**START**)：当SCL为高电平时，在SDA线上从高到低的跳变
- 停止条件(**STOP**)：当SCL为高电平时，在SDA线上从低到高的跳变

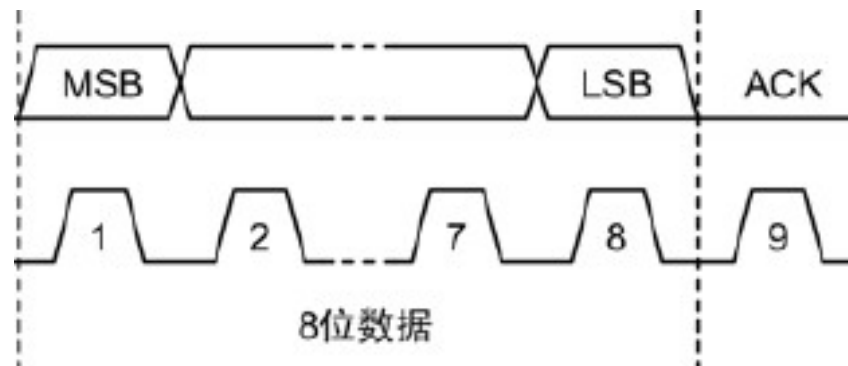
总线在起始条件之后被看作为忙状态；总线在停止条件之后被看作为空闲





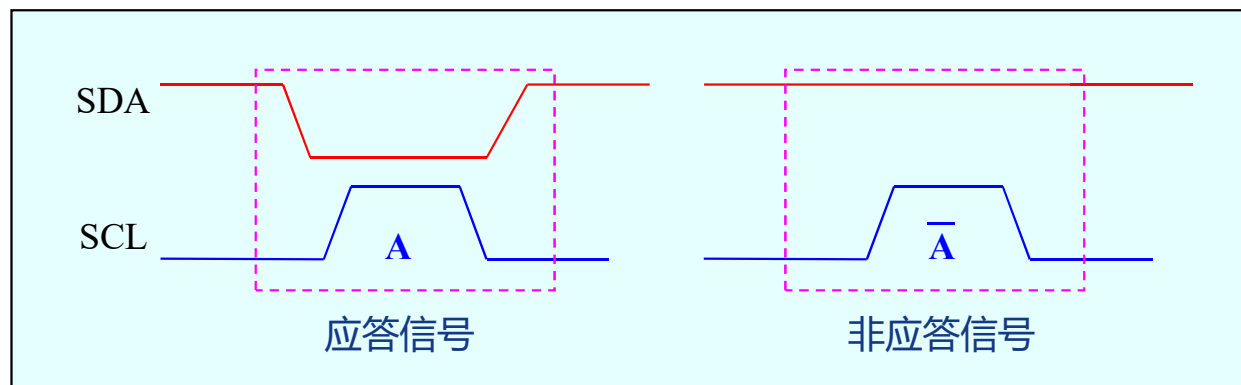
■ 字节格式

- SDA线上的每个字节必须为8位长，不限制每次传输的字节数
- 每个字节后面必须带有一个应答位
- ※ 当接收器不能接受另一个完整的字节时，它可以将时钟线SCL拉到低电平，以迫使发送器进入等待状态。当接收器释放时钟SCL时继续进行数据传输
- 数据传输时高位(MSB)在前，低位(LSB)在后



■ 应答和非应答信号(ACK)

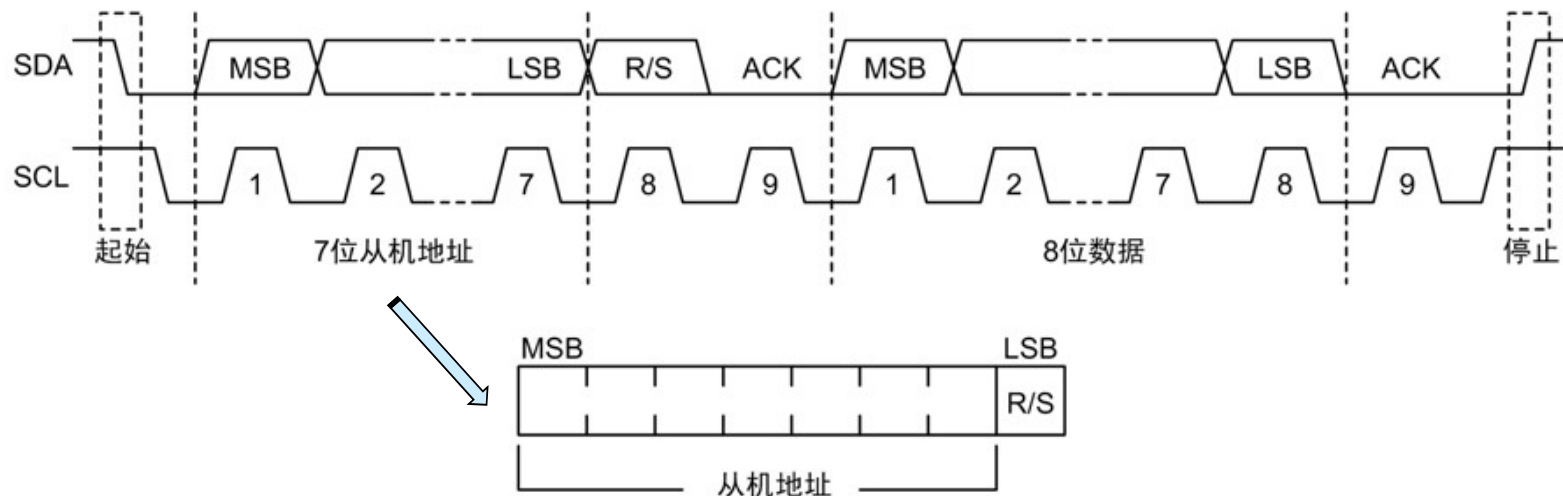
- SDA线上传输的每个字节后面必须带一个**应答位(A)**，由接收器在应答时钟脉冲期间拉低SDA形成
- 当主机作为接收器时，如果要结束通信，将在停止位之前发送**非应答信号(\bar{A})**





■ 7位从机地址的数据格式

- 起始信号后传送的**第一字节**数据具有特别的意义，其前7位为**从机地址** (Slave Address)，第8位为**数据方向**位R/S，0表示主机发送（写），1表示主机接收（读）





■ 数据地址（也称子地址）

- 数据地址也像普通数据一样进行传输，区分传输的到底是要读写的数据地址还是要读写的数据要靠收发双方具体的逻辑约定
- ※ **从机地址**：器件在I2C总线上被主机寻址的地址
- ※ **数据地址**：器件内部不同部件或存储单元的编址

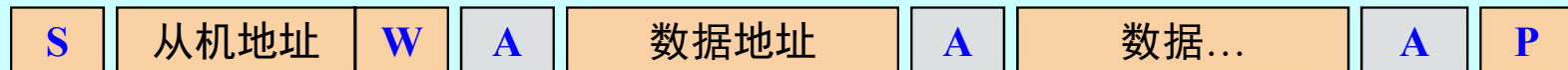
■ 重复起始

- 总线上的数据传输由主机产生的停止条件来终止
- 通过产生重复的起始条件和寻址另一个从机，主机可以继续总线上通信而无需先产生停止条件

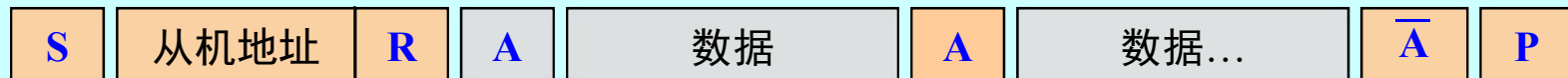


I2C总线操作模式

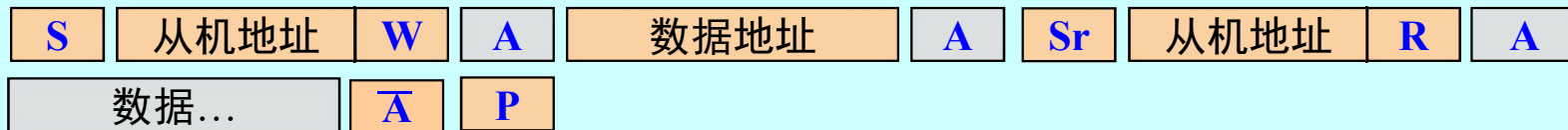
主发送（从接收）模式：





主接收（从发送）模式：当前数据地址



主接收（从发送）模式：指定数据地址



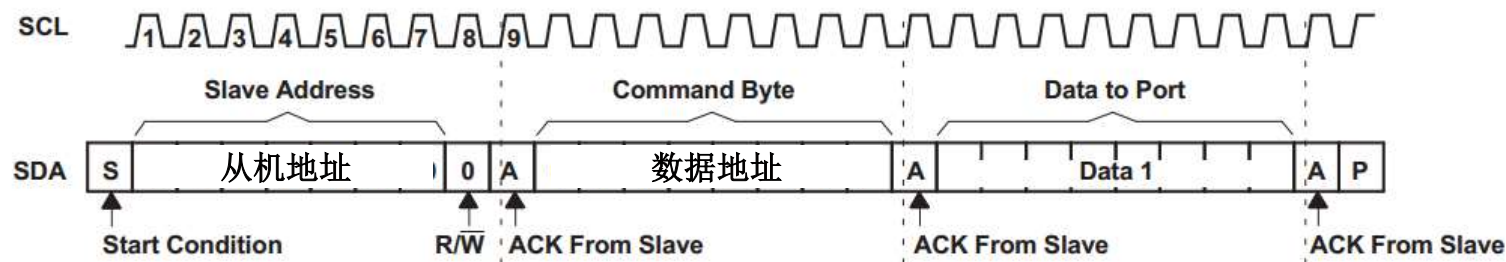
 主机发出
 从机发出

A = 应答
 \overline{A} = 非应答

S = 起始信号
P = 停止信号

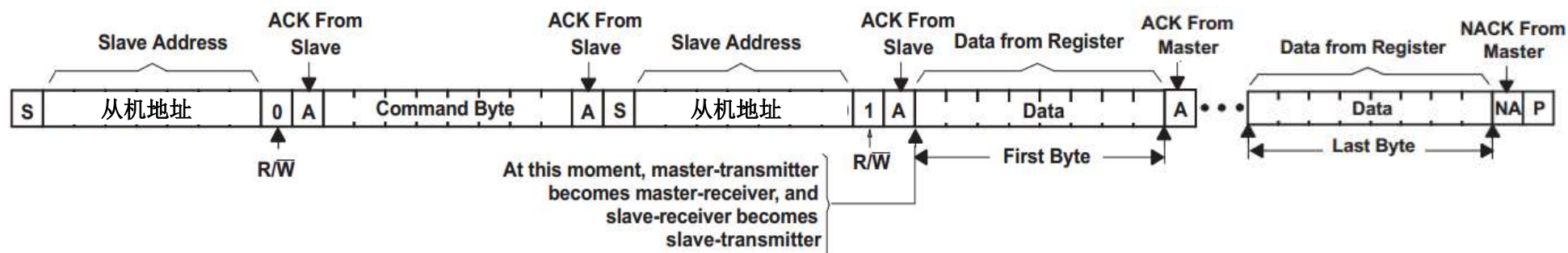
Sr = 重复起始信号

■ 例1：I2C主发送模式的操作过程（写操作）



- 第一个字节：7位**从机地址** + 1位**数据方向R/W** (0)
- 第二个字节：命令字节，8位**数据地址**给出写操作的起始地址
- 第3~N个字节：发送的**数据**，每个字节8位

■ 例2：I2C主接收模式的操作过程（读操作）

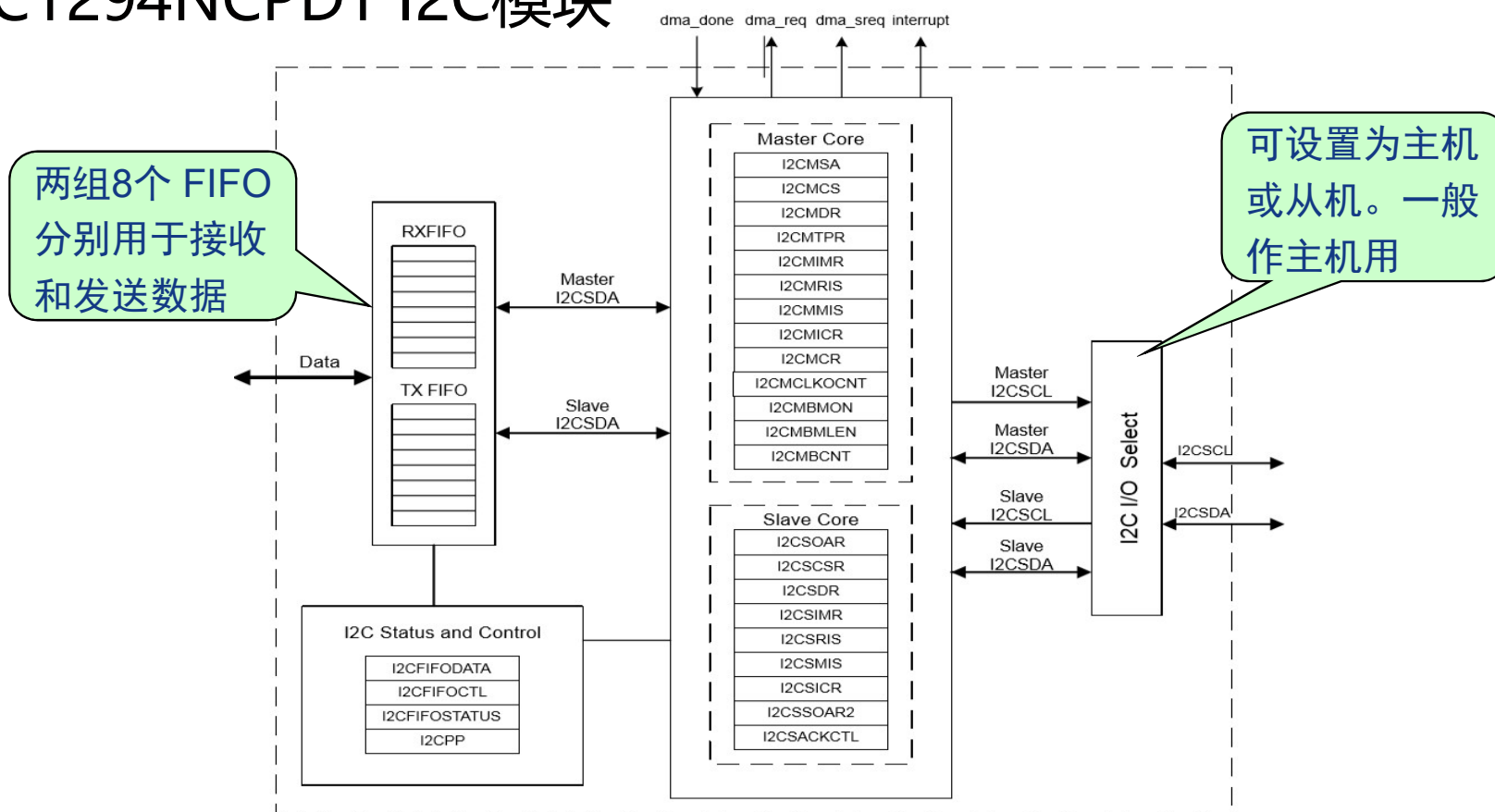


- 第一个字节：7位**从机地址** + 1位**数据方向R/W** (0)
- 第二个字节：命令字节， 8位**数据地址**给出读操作的起始地址
- 第三个字节：重启命令， 7位**从机地址** + 1位**数据方向R/W** (1)
- 第4~N个字节：接收的**数据**， 每个字节8位

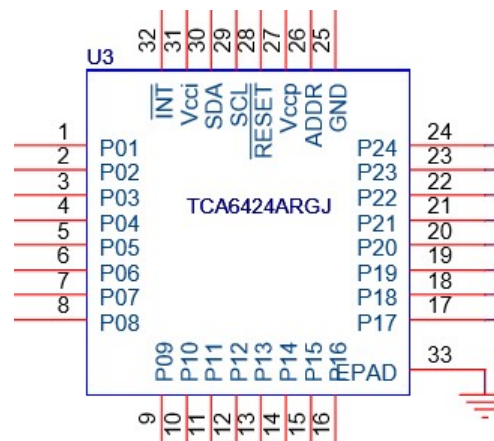
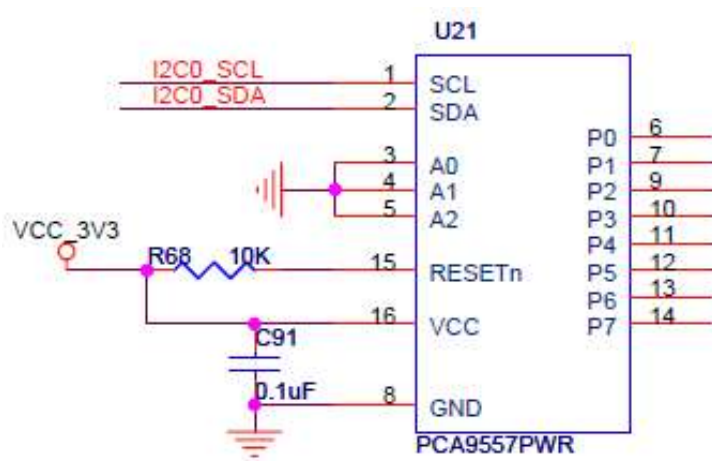


S800实验板上的I2C资源

■ TM4C1294NCPDT I2C模块



- S800实验板上的I2C资源
 - I2C转8位I/O扩展芯片PCA9557PWR
 - I2C转24位I/O扩展芯片TCA6424

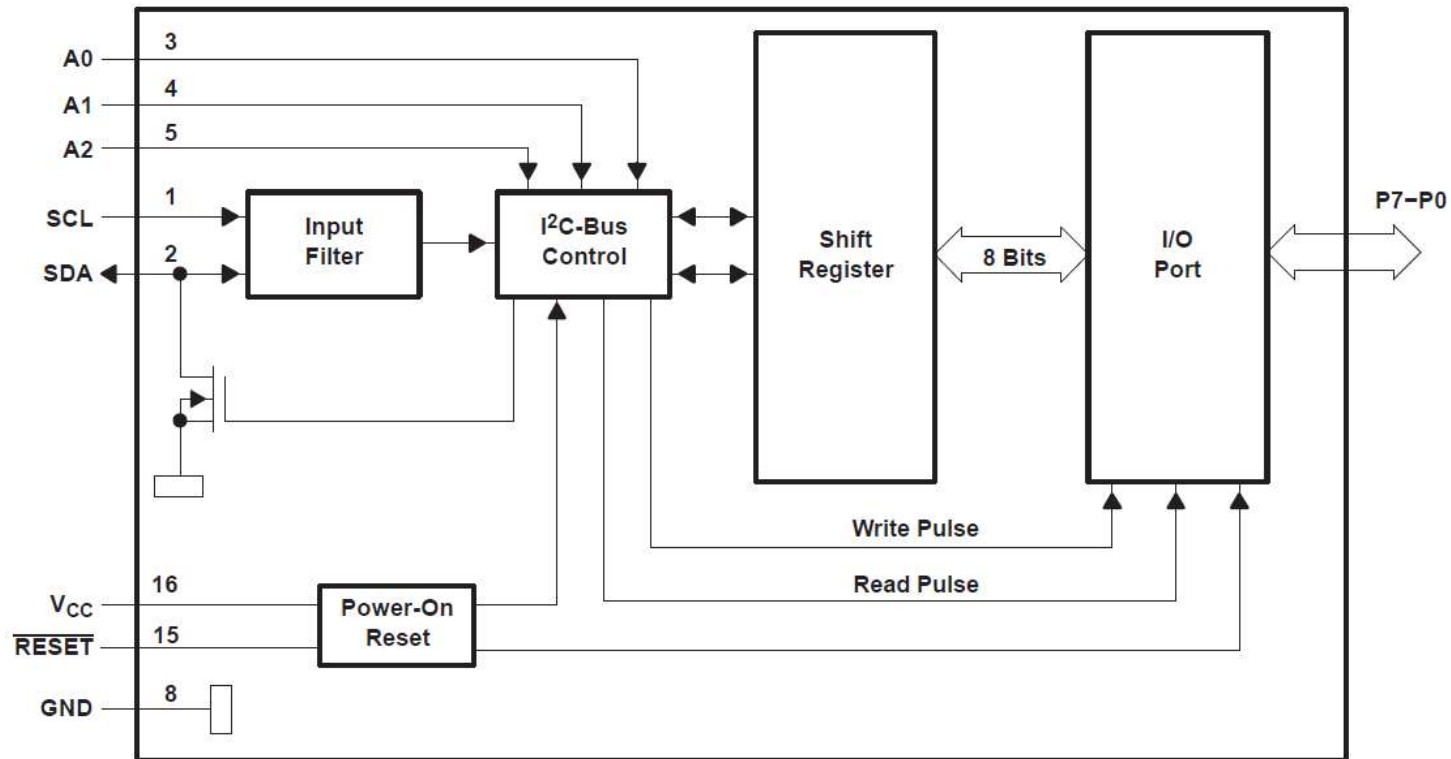




3.2.3 I²C转I/O扩展芯片PCA9557PWR

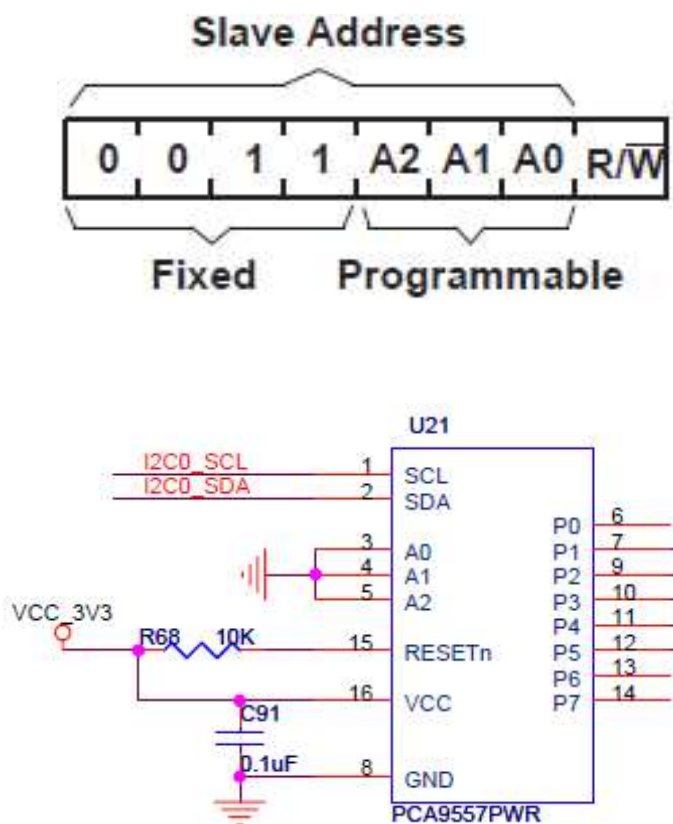
- PCA9557PWR芯片
 - REMOTE 8-BIT I2C AND SMBus LOW-POWER I/O EXPANDER
 - I2C to 8-BIT Parallel Port Expander
 - 400-kHz Fast I2C Bus
 - Three Hardware Address Pins Allow for Use of up to Eight Devices on I2C/SMBus
 - Noise Filter on SCL/SDA Inputs
 - 工作电压 2.3V ~ 5.5V

■ PCA9557PWR功能框图



- A. Pin numbers shown are for the D, DB, DGV, PW, and RGY packages.
- B. All I/Os are set to inputs at reset.

■ 从机地址格式



Address Reference

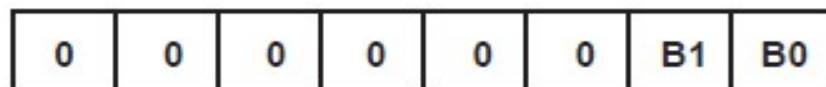
INPUTS			I ² C BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	24 (decimal), 18 (hexadecimal)
L	L	H	25 (decimal), 19 (hexadecimal)
L	H	L	26 (decimal), 1A (hexadecimal)
L	H	H	27 (decimal), 1B (hexadecimal)
H	L	L	28 (decimal), 1C (hexadecimal)
H	L	H	29 (decimal), 1D (hexadecimal)
H	H	L	30 (decimal), 1E (hexadecimal)
H	H	H	31 (decimal), 1F (hexadecimal)

```
#define PCA9557_I2CADDR 0x18
```




■ 命令字节

- 在成功收到从机对从机地址的应答后，I2C主机将发送一个命令字节，给出要读写的内部寄存器的地址（即子地址或数据地址），该子地址通过数据寄存器发送。命令字节的格式为：



- PCA9557PWR内部有四个端口寄存器

CONTROL REGISTER BITS		COMMAND BYTE (HEX)	REGISTER	PROTOCOL	POWER-UP DEFAULT
B1	B0				
0	0	0x00	Input Port	Read byte	xxxx xxxx
0	1	0x01	Output Port	Read/write byte	0000 0000
1	0	0x02	Polarity Inversion	Read/write byte	1111 0000
1	1	0x03	Configuration	Read/write byte	1111 1111

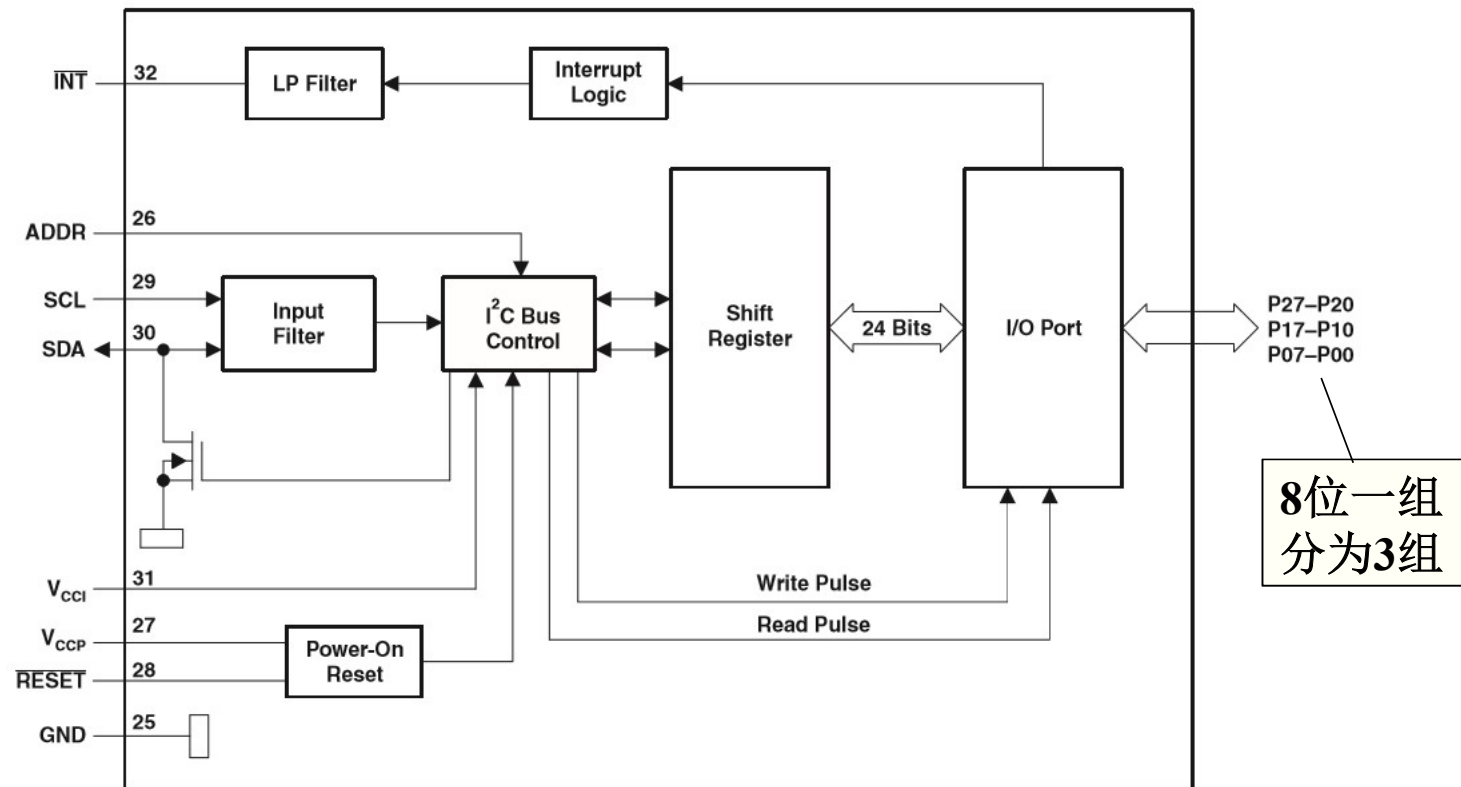
0为输出，
1为输入



3.2.3 I²C转I/O扩展芯片TCA6424

- TCA6424芯片
 - Low-Voltage 24-Bit I2C AND SMBus I/O Expander
 - I2C to 24-BIT Parallel Port Expander
 - 400-kHz Fast I2C Bus
 - Low Standby Current Consumption of 1 μ A
 - Schmitt-Trigger Action Allows Slow Input Transition and Better Switching Noise Immunity on SCL/SDA Inputs
 - **Latched Outputs** With High-Current Drive Maximum Capability for Directly Driving LEDs
 - 工作电压1.65V ~ 5.5V

■ TCA6424功能框图



■ 从机地址格式

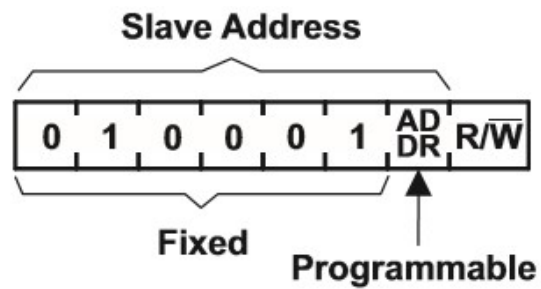
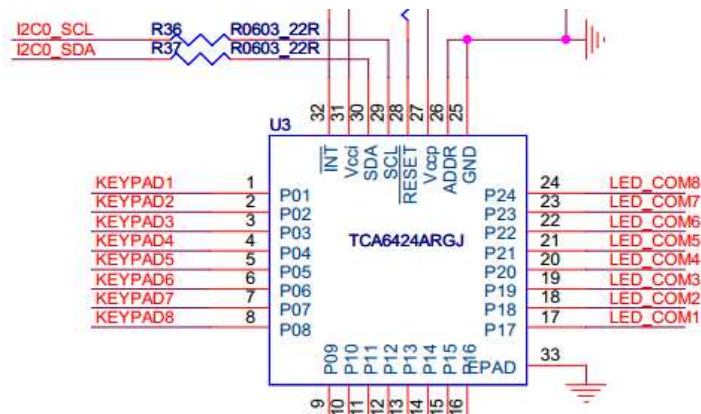


Table 3. Address Reference

ADDR	I ² C BUS SLAVE ADDRESS
L	34 (decimal), 22 (hexadecimal)
H	35 (decimal), 23 (hexadecimal)



```
#define PCA6424_I2CADDR 0x22
```



■ 命令字节格式

AI	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----

■ 3组端口共有12个内部寄存器

Table 4. Command Byte

CONTROL REGISTER BITS								AUTO-INCREMENT STATE	COMMAND BYTE (HEX)	REGISTER	PROTOCOL	POWER-UP DEFAULT
AI	B6	B5	B4	B3	B2	B1	B0					
0	0	0	0	0	0	0	0	Disable	00	Input Port 0	Read byte	xxxx xxxx ⁽¹⁾
1	0	0	0	0	0	0	0	Enable	80			
0	0	0	0	0	0	0	1	Disable	01	Input Port 1	Read byte	xxxx xxxx ⁽¹⁾
1	0	0	0	0	0	0	1	Enable	81			
0	0	0	0	0	0	1	0	Disable	02	Input Port 2	Read byte	xxxx xxxx ⁽¹⁾
1	0	0	0	0	0	1	0	Enable	82			
0	0	0	0	0	0	1	1	Disable	03	Reserved	Reserved	Reserved
1	0	0	0	0	0	1	1	Enable	83			
0	0	0	0	0	1	0	0	Disable	04	Output Port 0	Read/write byte	1111 1111
1	0	0	0	0	1	0	0	Enable	84			
0	0	0	0	0	1	0	1	Disable	05	Output Port 1	Read/write byte	1111 1111
1	0	0	0	0	1	0	1	Enable	85			
0	0	0	0	0	1	1	0	Disable	06	Output Port 2	Read/write byte	1111 1111
1	0	0	0	0	1	1	0	Enable	86			
0	0	0	0	0	1	1	1	Disable	07	Reserved	Reserved	Reserved
1	0	0	0	0	1	1	1	Enable	87			
0	0	0	0	1	0	0	0	Disable	08	Polarity Inversion Port 0	Read/write byte	0000 0000
1	0	0	0	1	0	0	0	Enable	88			
0	0	0	0	1	0	0	1	Disable	09	Polarity Inversion Port 1	Read/write byte	0000 0000
1	0	0	0	1	0	0	1	Enable	89			
0	0	0	0	1	0	1	0	Disable	0A	Polarity Inversion Port 2	Read/write byte	0000 0000
1	0	0	0	1	0	1	0	Enable	8A			
0	0	0	0	1	0	1	1	Disable	0B	Reserved	Reserved	Reserved
1	0	0	0	1	0	1	1	Enable	8B			
0	0	0	0	1	1	0	0	Disable	0C	Configuration Port 0	Read/write byte	1111 1111
1	0	0	0	1	1	0	0	Enable	8C			
0	0	0	0	1	1	0	1	Disable	0D	Configuration Port 1	Read/write byte	1111 1111
1	0	0	0	1	1	0	1	Enable	8D			
0	0	0	0	1	1	1	0	Disable	0E	Configuration Port 2	Read/write byte	1111 1111
1	0	0	0	1	1	1	0	Enable	8E			

0为输出,
1为输入



实验二 I2C扩展及SYSTICK中断实验

■ 实验内容

- 例程exp2-1.c: I2C和两个扩展芯片的初始化, PCA9557点亮所有LED; TCA6424控制数码管显示0在第一位。

■ 编程要点

1. 程序结构 (初始化+3个任务)
2. 初始化: I2C初始化配置
3. 任务1: PCA9557控制LED灯
4. 任务2: TCA6424控制数码管显示

```
int main(void)
{
    uint8_t result;

    S800_Clock_Init();
    S800_GPIO_Init();
    S800_I2C_Init();

    while (1)
    {
        //点数码管
        result = I2C0_WriteByte(TCA6424_I2CADDR, TCA6424_OUTPUT_PORT1, seg7[0]);
        result = I2C0_WriteByte(TCA6424_I2CADDR, TCA6424_OUTPUT_PORT2, (uint8_t)(1));

        //点LED灯
        result = I2C0_WriteByte(PCA9557_I2CADDR, PCA9557_OUTPUT, 0x0);

        //PF0闪烁 (Turn over the PF0)
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, ~GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0));
        SysCtlDelay(ui32SysClock/6); //延时0.5s
    }
}
```

初始化

task1

task2

task3





■ I2C模块初始化配置步骤 (**S800_I2C0_Init()**)

1. 使能I2C模块
 2. 使能提供I2C接口信号的GPIO端口
 3. 将GPIO引脚设置为I2C复用功能
 4. 配置并使能I2C主模式
- 通过系统外设控制函数设置
- 通过GPIO控制函数设置
- 通过I2C控制函数设置

I2C接口信号

I2C接口信号由GPIO引脚的复用功能提供

※ GPIO引脚功能

- 数字输入/输出：控制按键、LED灯等
- 模拟输入：ADC的输入等
- 复用功能：I2C、UART、USB、SSI、CAN等

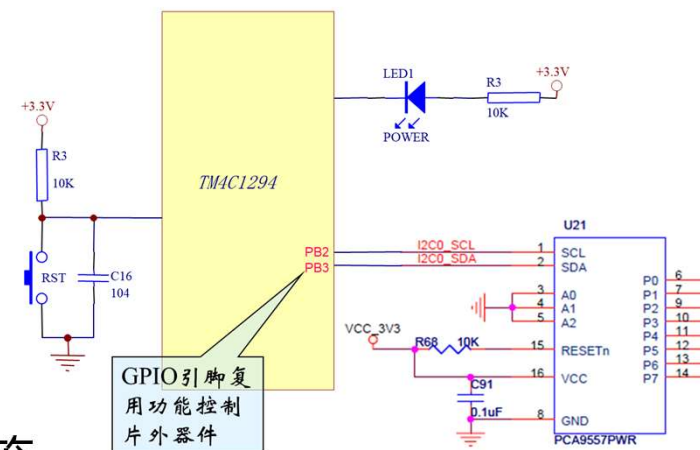


Table 18-1. I2C Signals (128TQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
I2C0SCL	91	PB2 (2)	I/O	OD	I ² C module 0 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I2C0SDA	92	PB3 (2)	I/O	OD	I ² C module 0 data.



- TivaWare的I2C库函数
 - 驱动程序参见driverlib/i2c.c，API定义在driverlib/i2c.h
- I2C主机模式初始化设置
 - I2CMasterInitExpClk(): 配置I²C模块为主机模式，并选择时钟和通信速率（100Kbps或400Kbps）
 - I2CMasterEnable(): 使能主机模式

```
void I2CMasterInitExpClk (uint32_t ui32Base, uint32_t ui32I2CCLK, bool bFast);
```

```
void I2CMasterEnable (uint32_t ui32Base);
```

其中：ui32Base：I2C模块基地址， ui32I2CCLK：时钟频率

bFast：true表示传输率 400Kbps， false为100Kbps



1. 使能I2C0模块 (调用系统外设控制函数)

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0)
```

2. 使能提供I2C接口信号的GPIOB端口

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB)
```

I2C模块名和GPIO端口名都定义在sysctl.h中

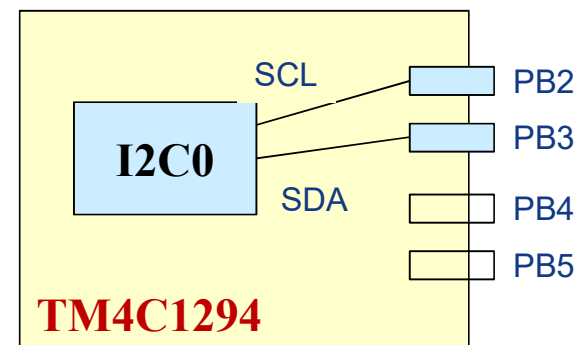
3. 将GPIO引脚PB2, PB3设置为I2C复用功能

```
GPIOPinConfigure(GPIO_PB2_I2C0SCL)
```

```
GPIOPinConfigure(GPIO_PB3_I2C0SDA)
```

```
GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2)
```

```
GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3)
```



4. 配置并使能I2C0主模式

```
I2CMasterInitExpClk(I2C0_BASE, ui32SysClock, true); //400Kbps
```

```
I2CMasterEnable(I2C0_BASE);
```

详见S800_I2C0_Init()



实验二 I2C扩展及SYSTICK中断实验

■ 实验内容

- 例程exp2-1.c: I2C和两个扩展芯片的初始化, PCA9557点亮所有LED; TCA6424控制数码管显示0在第一位。

■ 编程要点

1. 程序结构 (初始化+3个任务)
2. 初始化: I2C初始化配置
3. 任务1: PCA9557控制LED灯
4. 任务2: TCA6424控制数码管显示

```
int main(void)
{
    uint8_t result;

    S800_Clock_Init();           初始化
    S800_GPIO_Init();
    S800_I2C0_Init();

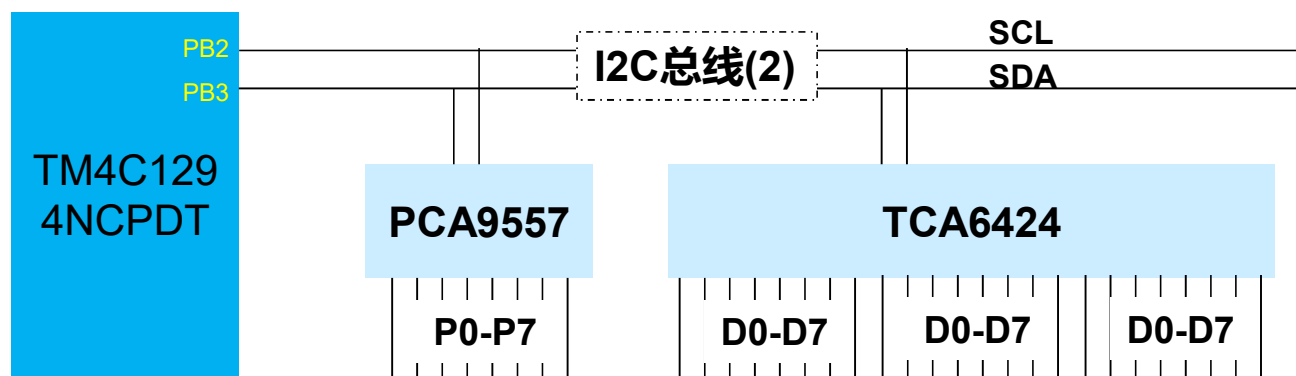
    while (1)
    {
        //点数码管
        result = I2C0_WriteByte(TCA6424_I2CADDR, TCA6424_OUTPUT_PORT1, seg7[0]);
        result = I2C0_WriteByte(TCA6424_I2CADDR, TCA6424_OUTPUT_PORT2, (uint8_t)(1));
        task1

        //点LED灯
        result = I2C0_WriteByte(PCA9557_I2CADDR, PCA9557_OUTPUT, 0x0);
        task2

        //PF0闪烁 (Turn over the PF0)
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, ~GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0));
        SysCtlDelay(ui32SysClock/6); //延时0.5s
        task3
    }
}
```



- I2C扩展芯片PCA9557PWR和TCA6424的控制
 - I2C控制引脚: **PB2 – I2C0SCL**, **PB3 – I2C0SDA**
 - PCA9557PWR的初始化配置和读写
 - TCA6424的初始化配置和读写
- } 通过I2C总线操作设置





- TivaWare的I2C主机模式总线操作控制函数
 - I2CMasterSlaveAddrSet(): 设置从机地址和读写控制位
 - I2CMasterDataPut()、I2CMasterDataGet(): 将发送的数据送输出寄存器、或者从输入寄存器读取数据
 - I2CMasterControl(): 启动主模式下收发数据的总线动作

```
void I2CMasterSlaveAddrSet ( uint32_t ui32Base, uint8_t ui8SlaveAddr, bool bReceive );  
void I2CMasterDataPut ( uint32_t ui32Base, uint8_t ui8Data );  
void I2CMasterControl ( uint32_t ui32Base, uint32_t ui32Cmd );
```

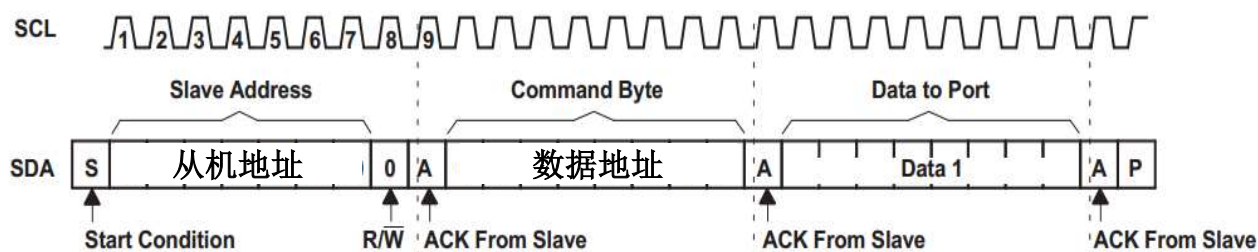
其中： ui32Base: I2C模块基地址， ui8SlaveAddr: 从机地址
bReceive: **true**表示读操作， **false**为写操作
ui8Data: 发送的数据， ui32Cmd: 发送的[控制命令](#)



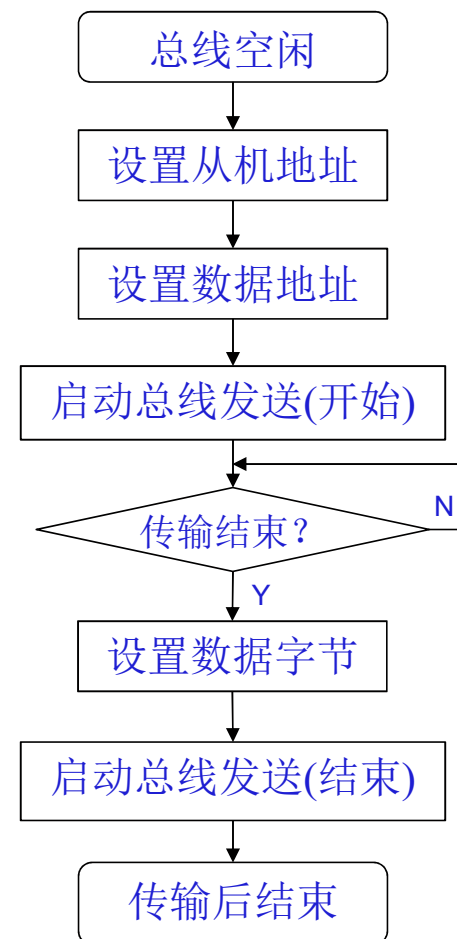
■ I2C主机控制命令 (参见driverlib\i2c.h)

#define I2C_MASTER_CMD_SINGLE_SEND	0x00000007
#define I2C_MASTER_CMD_SINGLE_RECEIVE	0x00000007
#define I2C_MASTER_CMD_BURST_SEND_START	0x00000003
#define I2C_MASTER_CMD_BURST_SEND_CONT	0x00000001
#define I2C_MASTER_CMD_BURST_SEND_FINISH	0x00000005
#define I2C_MASTER_CMD_BURST_SEND_ERROR_STOP	0x00000004
#define I2C_MASTER_CMD_BURST_RECEIVE_START	0x0000000b
#define I2C_MASTER_CMD_BURST_RECEIVE_CONT	0x00000009
#define I2C_MASTER_CMD_BURST_RECEIVE_FINISH	0x00000005
#define I2C_MASTER_CMD_BURST_RECEIVE_ERROR_STOP	0x00000004

■ I2C总线操作编程——写单字节数据 (主发送模式)



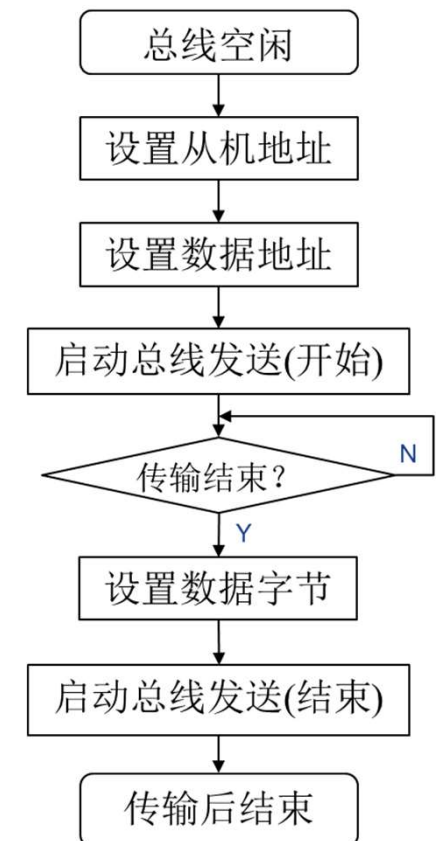
- 程序流程图 (省略了出错检测)
- 可以采用查询方式或者中断方式



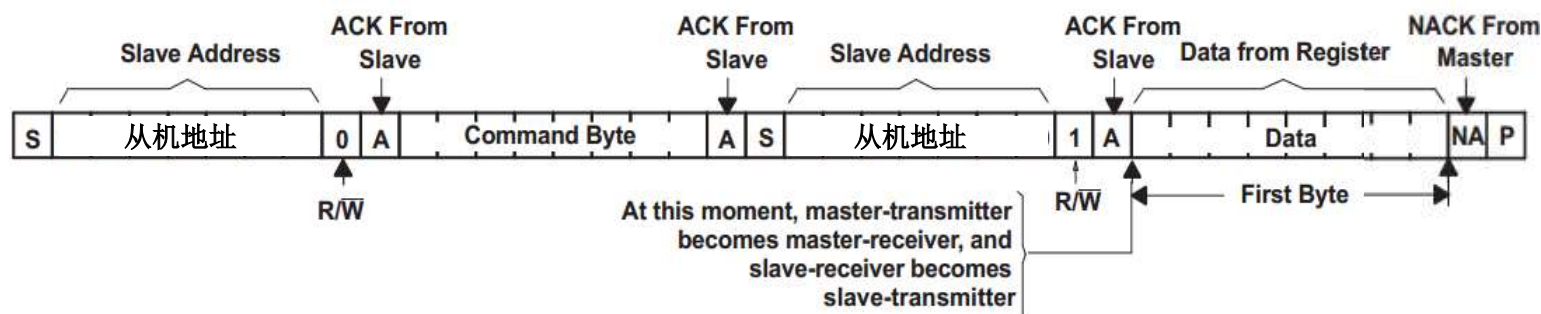


uint8_t I2C0_WriteByte(uint8_t DevAddr, uint8_t RegAddr, uint8_t WriteData) //写操作函数

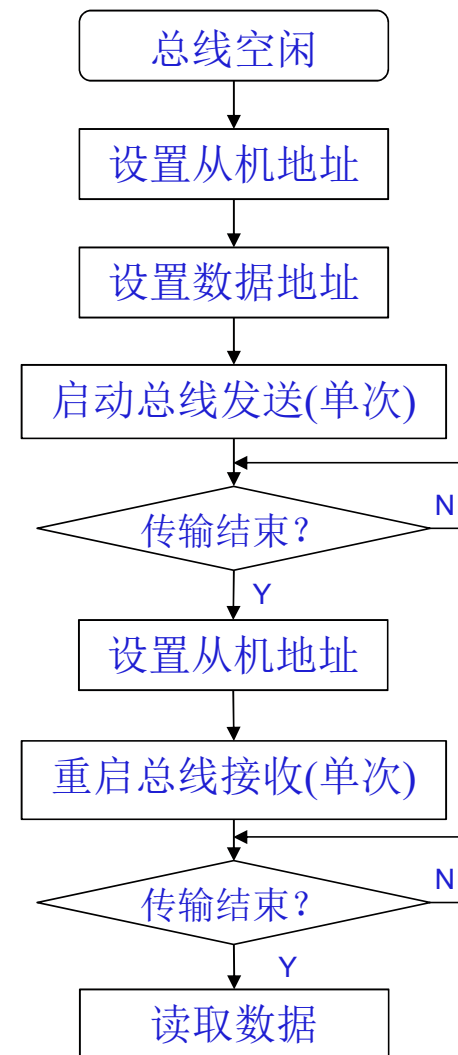
```
{
    uint8_t rop;
    while (I2CMasterBusy(I2C0_BASE)){}; //遇忙等待
    I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false); //设置从机地址, 写
    I2CMasterDataPut(I2C0_BASE, RegAddr); //设置数据地址 (命令字节)
    //启动"总线发送开始"
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE)){}; //等待传输结束
    rop = (uint8_t)I2CMasterErr(I2C0_BASE); //检测错误, 0-无错
    I2CMasterDataPut(I2C0_BASE, WriteData); //设置数据字节
    //启动"总线发送后结束"
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(I2C0_BASE)){};
    rop = (uint8_t)I2CMasterErr(I2C0_BASE); //检测错误, 0-无错
    return rop;
}
```



■ I2C总线操作编程——读单字节数据 (主接收模式)

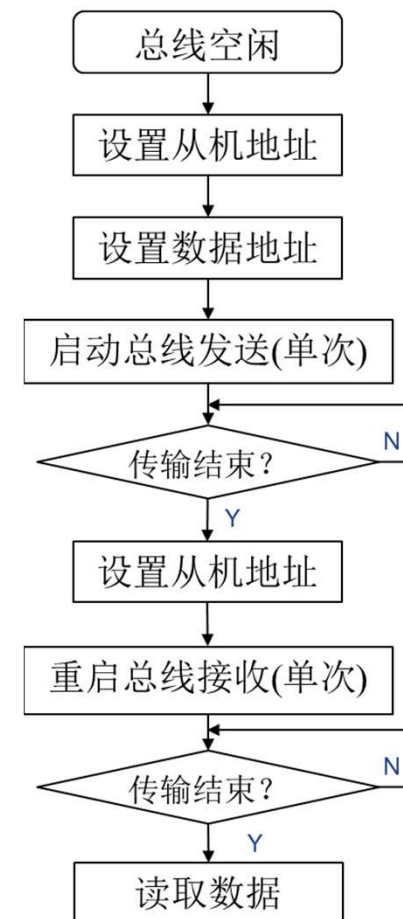


- 程序流程图 (省略了出错检测)
- 可以采用查询方式或者中断方式



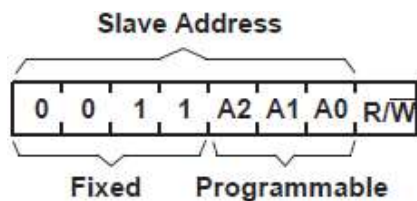


```
uint8_t I2C0_ReadByte(uint8_t DevAddr, uint8_t RegAddr) //读操作函数
{
    uint8_t value, rop;
    while(I2CMasterBusy(I2C0_BASE)){}; //遇忙等待
    I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false); //设置从机地址, 写
    I2CMasterDataPut(I2C0_BASE, RegAddr); //设置数据地址
    //启动"总线发送"
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
    while(I2CMasterBusBusy(I2C0_BASE)) {};
    rop = (uint8_t)I2CMasterErr(I2C0_BASE);
    I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, true); //设置从机地址, 读
    //重启 "总线接收"
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
    while(I2CMasterBusBusy(I2C0_BASE)) {};
    value = I2CMasterDataGet(I2C0_BASE); //读取数据
    return value;
}
```



■ 任务1：PCA9557PWR控制LED灯

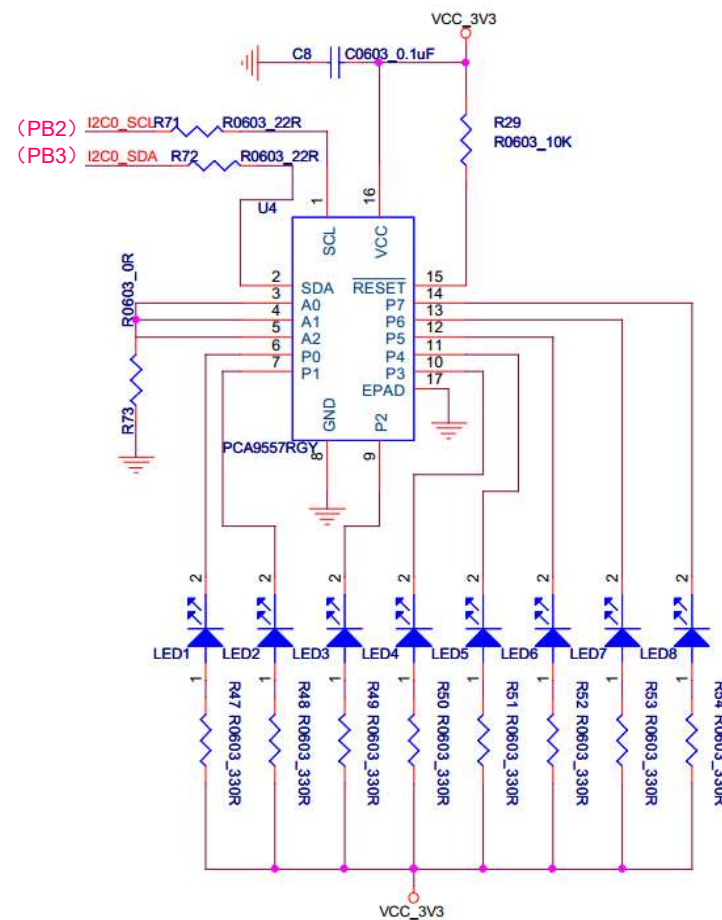
- 从机地址：A2=A1=A0=0，从机地址=0x18



- 数据地址：四个端口，地址为0x00~0x03

定义从机地址和数据地址：

```
#define PCA9557_I2CADDR    0x18
#define PCA9557_INPUT      0x00
#define PCA9557_OUTPUT     0x01
#define PCA9557_POLINVERT  0x02
#define PCA9557_CONFIG     0x03
```



1. PCA9557PWR的初始化配置

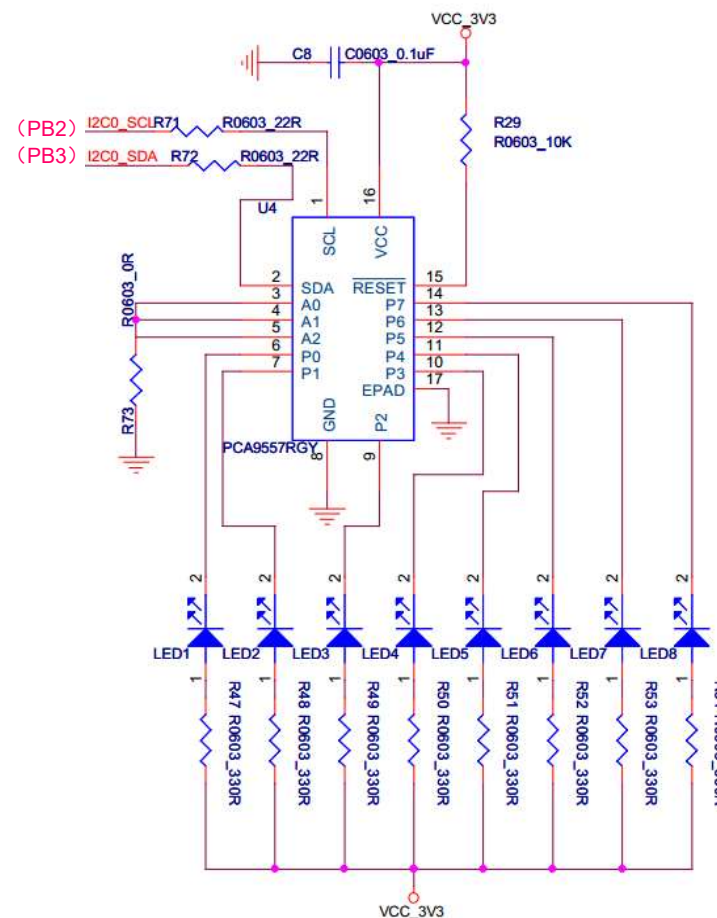
- Configuration端口（数据地址0x03）用于引脚P0-P7的输入/输出设置
- 通过I2C写操作，往Configuration端口送0x0，将P0-P7配置为输出

```
I2C0_WriteByte(PCA9557_I2CADDR,  
PCA9557_CONFIG, 0x00);
```

2. PCA9557PWR控制LED灯的亮灭

- 往PCA9557的Output端口送数据以控制LED灯的亮灭，低电平点灯。如：

```
I2C0_WriteByte(PCA9557_I2CADDR,  
PCA9557_OUTPUT, ~(0x01<<2));
```





■ 例：LED跑马灯（流水灯）

```
int main(void)
{
    volatile uint8_t cnt=0;
    .....
    while (1) {
        //点LED灯
        I2C0_WriteByte(PCA9557_I2CADDR, PCA9557_OUTPUT, ~(1<<cnt));
        cnt = (cnt+1) % 8;
        SysCtlDelay(ui32SysClock/6); //延时0.5s
    }
}
```



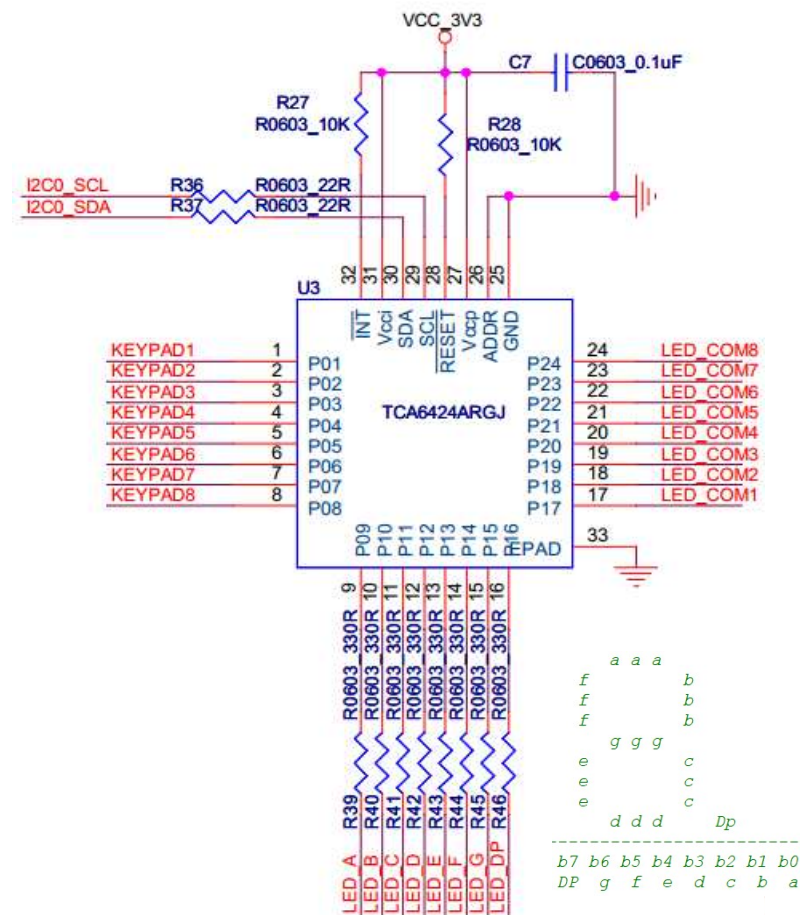

■ 任务2：TCA6424控制数码管显示

- 从机地址：ADDR=0，从机地址=0x22
- 数据地址：3组端口12个端口寄存器

定义从机地址和数据地址：

```
#define PCA6424_I2CADDR      0x22

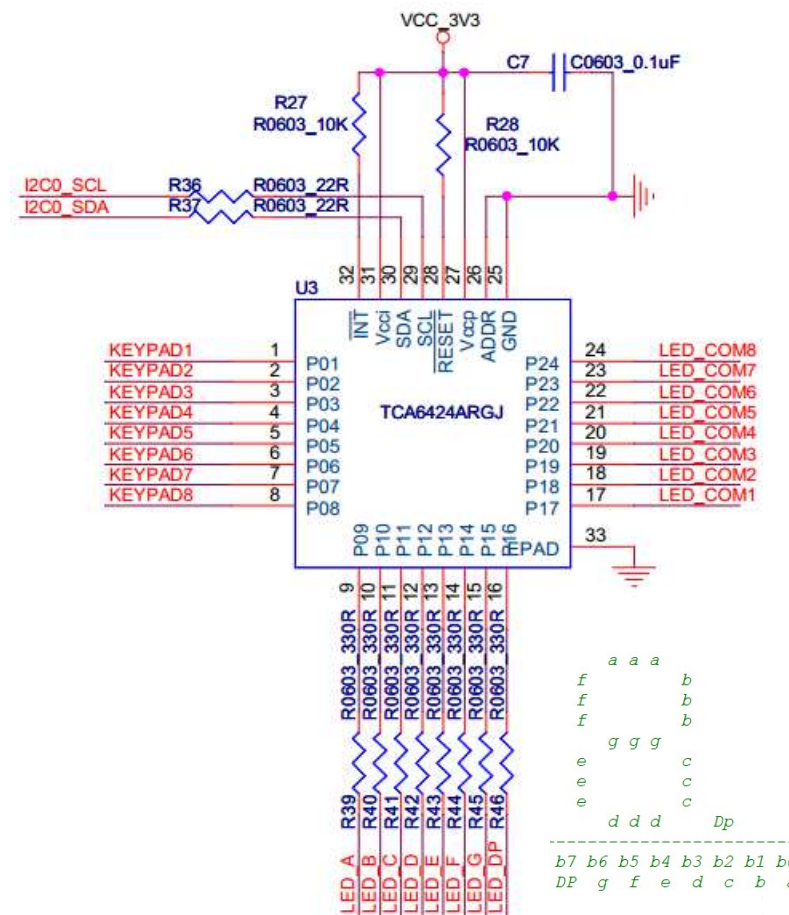
#define TCA6424_CONFIG_PORT0 0x0c
#define TCA6424_CONFIG_PORT1 0x0d
#define TCA6424_CONFIG_PORT2 0x0e
#define TCA6424_INPUT_PORT0  0x00
#define TCA6424_INPUT_PORT1  0x01
#define TCA6424_INPUT_PORT2  0x02
#define TCA6424_OUTPUT_PORT0 0x04
#define TCA6424_OUTPUT_PORT1 0x05
#define TCA6424_OUTPUT_PORT2 0x06
```





■ 任务2：TCA6424控制数码管显示

- P0口 (P01-08) : 8位, 输入, 接蓝板上按键SW1-SW8
- P1口 (P09-16) : 8位, 输出, 接蓝板上8个动态共阴数码管的脚位信号, 高电平时点亮相应的笔划
- P2口 (P17-24) : 8位, 输出, 接蓝板上8个动态共阴数码管的片选信号, 高电平时导通三极管, 从而选通对应的数码位



1. 初始化TCA6424将 P0口配置为输入， P1和P2口配置为输出

```
I2C0_WriteByte(TCA6424_I2CADDR,  
                TCA6424_CONFIG_PORT0,0xff); //设为输入
```

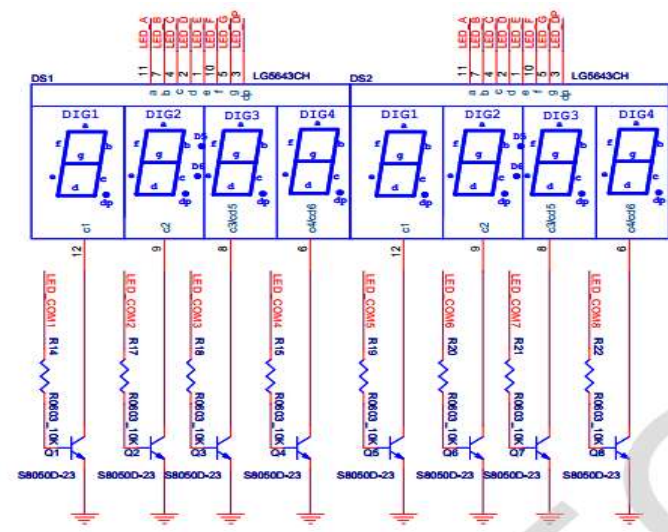
```
I2C0_WriteByte(TCA6424_I2CADDR,  
                TCA6424_CONFIG_PORT1,0x00); //设为输出
```

```
I2C0_WriteByte(TCA6424_I2CADDR,  
                TCA6424_CONFIG_PORT2,0x00); //设为输出
```

2. 数码管显示：往P1口的Output寄存器送字模， 往P2口的Output寄存器送选中的数码位

```
I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, 0x3f); // P1口送字模'0'
```

```
I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, 0x01<<2); //点亮第3个码管
```





■ 例：数码管和LED的跑马灯

```
uint8_t seg7[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,...};
```

```
int main(void)
```

```
{  
    volatile uint8_t cnt=0;  
    .....  
    while (1) {  
        //点数码管  
        I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[cnt+1]);  
        I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,(uint8_t)(1<<cnt));  
        //点LED灯  
        I2C0_WriteByte(PCA9557_I2CADDR, PCA9557_OUTPUT, ~(1<<cnt));  
        cnt = (cnt+1) % 8;  
        SysCtlDelay(ui32SysClock/6); //延时0.5s  
    }  
}
```

➤ 数码管同时显示多个字符的要点

- 1) 防拖影：先往P2口写0，再给码管值和位选
- 2) 防频闪：循环刷新 + 控制延时

■ TCA6424按键读取（蓝板上的SW1~8）

- 读P0口的Input寄存器，按下为低电平

```
uint8_t key;
```

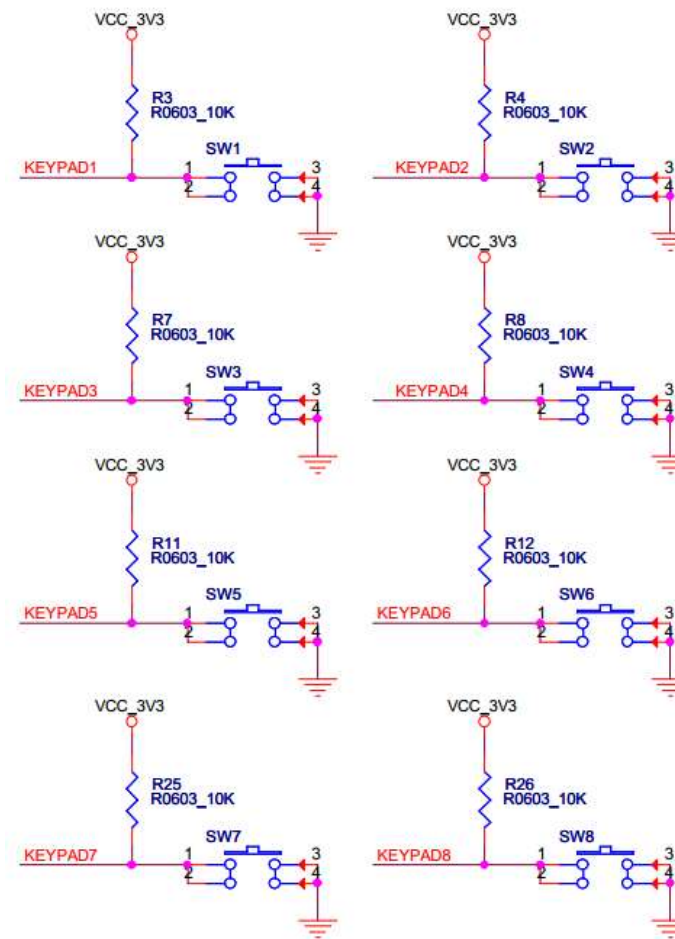
```
key = ~I2C0_ReadByte(TCA6424_I2CADDR,  
                      TCA6424_INPUT_PORT0);
```

```
if (key & 0x01) //SW1按下
```

```
...
```

```
if (key & 0x02) //SW2按下
```

```
...
```





实验二 I2C扩展及SYSTICK中断实验

■ 实验内容

- 例程exp2-3.c, 利用SysTick定时器实现实验2-2, 并控制数码管和LED跑马灯的频率为500ms, PF0闪烁频率为50ms。

■ 编程要点

1. 程序结构（前后台程序）
2. SysTick定时器控制多任务调度

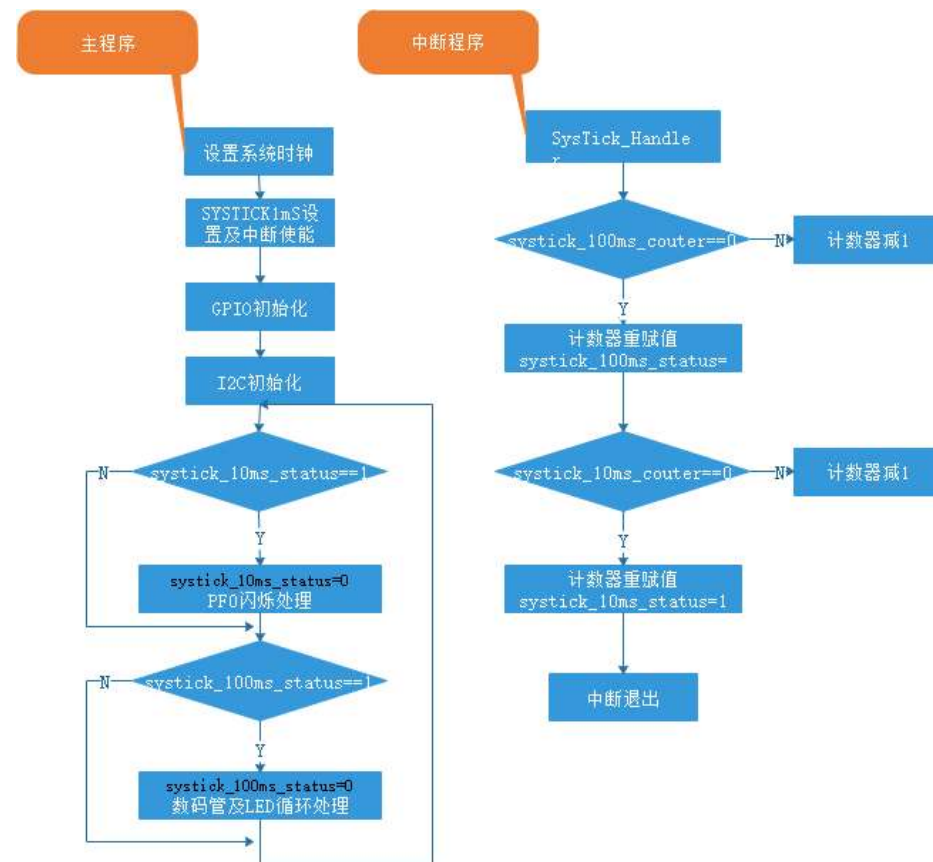


前后台程序

- SysTick设置基本定时1ms（时间片）
- 根据需要定义软件计时器

```
void SysTick_Handler(void)
{
    if (systick_100ms_couter == 0) { //100ms计时器
        systick_100ms_couter = 100; //全局变量
        systick_100ms_status = 1;    //全局变量
    }
    else systick_100ms_couter--;

    if (systick_10ms_couter == 0) { //10ms计时器
        systick_10ms_couter = 10;    //全局变量
        systick_10ms_status = 1;     //全局变量
    }
    else systick_10ms_couter--;
}
```





- 完成实验二，提交实验报告和源程序 -