

# IIC 通信



## ➤ 内容概要:

- I<sup>2</sup>C总线的基本概念
- I<sup>2</sup>C总线的通讯
- TM4C1294的I<sup>2</sup>C模块的功能
- TM4C1294的I<sup>2</sup>C模块的使用
- TM4C1294的I<sup>2</sup>C模块的应用



- I<sup>2</sup>C是Inter-Integrated Circuit的简称
  - 读作：I-squared-C
- I<sup>2</sup>C为了让主板、嵌入式系统或手机用以连接低速周边外部设备而发展。

I<sup>2</sup>C总线广泛应用在EEPROM、实时时钟、LCD、及其他芯片的接口。  
I<sup>2</sup>C允许相当大的工作电压范围，典型的电压基准为：**+3.3V**或**+5V**。

- I<sup>2</sup>C总线是由Philips公司开发的一种简单、双向二线制同步串行总线。
- 它只需要两根线即可在连接于总线上的器件之间传送信息。



I<sup>2</sup>C 总线具有如下特性:

## 1)总线协议简单容易

实现协议的基本部分相当简单,因此在芯片内部以硬件的方法实现 I<sup>2</sup>C 部件的逻辑是容易的。即使 MCU 内部没有硬件的 I<sup>2</sup>C 总线接口,也能够方便的利用开漏的I/O (如果没有,可用准双向I/O代替)来模拟实现。

## 2)支持的器件多主流半导体公司生产的大量器件

带有 I<sup>2</sup>C 器件提供了广阔的空间。在现代微控制器设计当中I<sup>2</sup>C 总线接口已经成为标准的重要片内外设之一。

## 3)总线上可同时挂接多个器件

同一条 I<sup>2</sup>C 总线可以挂接多个器件,器件之间靠不同的编址来区分的,不需要附加的I/O 线或地址译码部件。

## 4)总线可裁减性好

在原有总线连接的基础上可以随时新增或者删除器件。用软件可以容易实现 I<sup>2</sup>C 线的自检功能,能够及时发现总线上的变动。



## 5) 总线电气兼容性

I<sup>2</sup>C 总线规定器件之间以开漏 I/O 相连接，只要选取适当的上拉电阻即能实现不同逻辑电平之间的互联通信，而不需要额外的转换。

## 6) 支持多种通信方式

一主多从、多主机通信及广播模式等。

## 7) 通信速率高并兼顾低速通信

I<sup>2</sup>C 总线标准传输速率为 100kbps（每秒100K 位）。在快速模式下为 400kbps。按照后来修订的版本，位速率可高达3.4Mbps。

I<sup>2</sup>C总线的通信速率也可以低至几kbps 以下，用以支持低速器件（比如软件模拟的实现）或者用来延长通信距离。从机也可以再接受和相应一个字节后使 SCL 线保持低电平迫使主机进入等待状态直到从机准备好下一个要传输的字节。

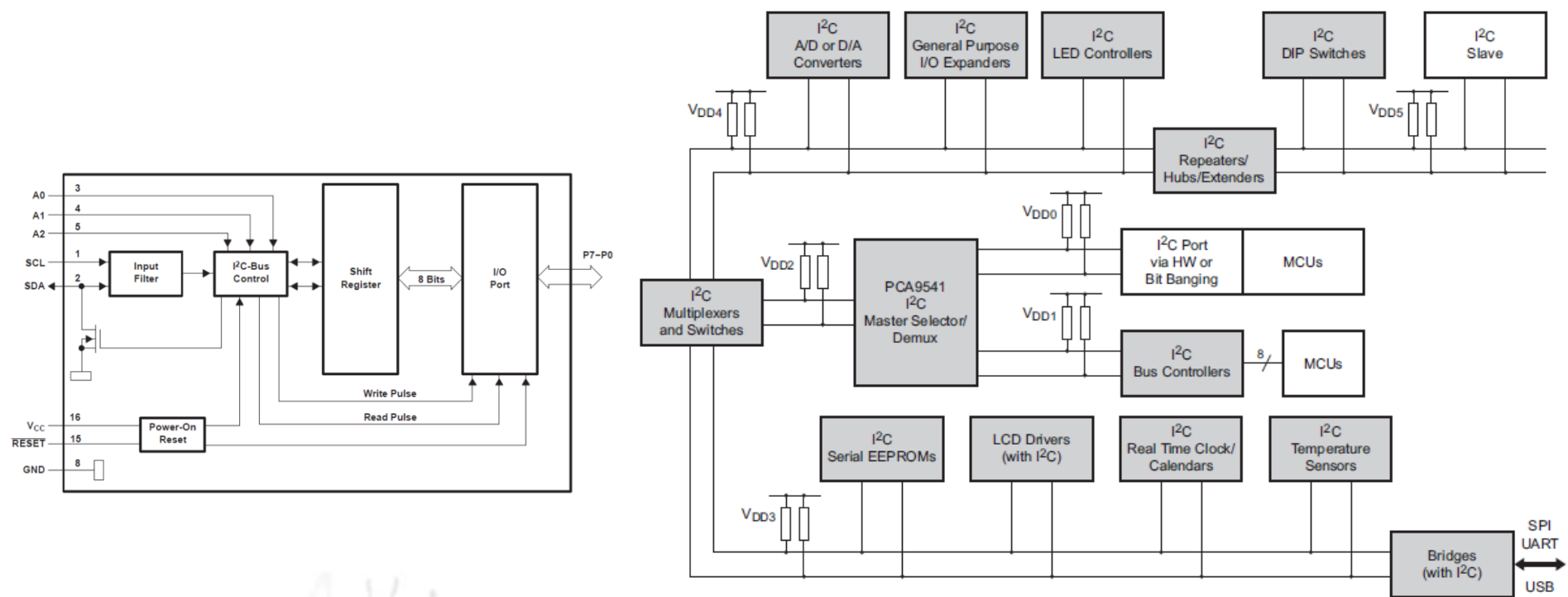
## 8) 有一定的通信距离

I<sup>2</sup>C 总线通信距离通常为几米到十几米。通过降低传输速率、屏蔽、中断方法，通信距离也可延长到数十米乃至数百米以上。



## ➤ I<sup>2</sup>C总线结构

- I<sup>2</sup>C的两根线SDA（串行**数据**线）和SCL（串行**时**钟线）都是双向I/O线，接口电路为**开漏输出**，需通过**上拉电阻**接电源VCC，当总线**空闲**时，两根线都是**高电平**。
- I<sup>2</sup>C总线是一种**多主控**总线，即可以在总线上放置多个主设备节点，在停止位（P）发出后，即通讯结束后，主设备节点可以成为从设备节点。



002aac858

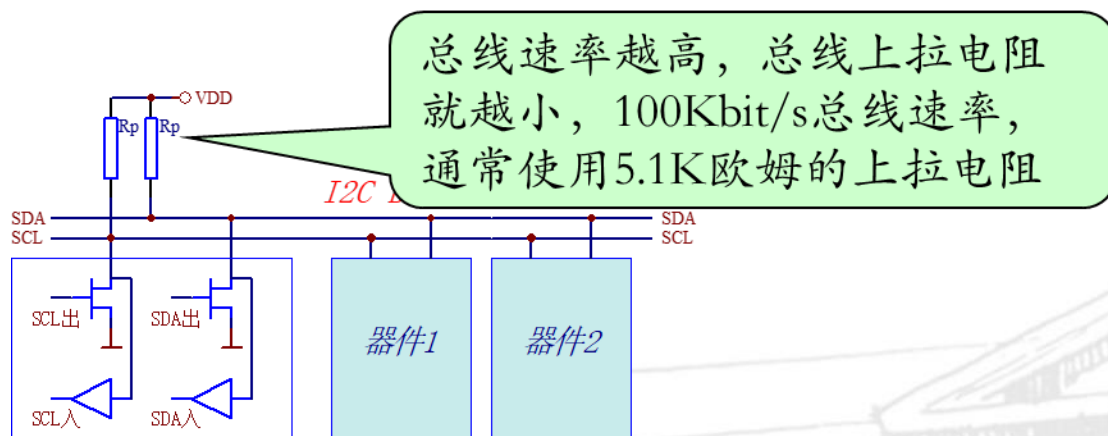
Fig 1. Example of I<sup>2</sup>C-bus applications

- I<sup>2</sup>C通信双方地位不对等，通信由主设备发起，并主导传输过程，从设备按I<sup>2</sup>C协议接收主设备发送的数据，并及时给出响应。
- 一个节点主设备、从设备由通信双方决定（I<sup>2</sup>C协议本身无规定），既能当主设备，也能当从设备（需要软件进行配置）。
- 主设备负责调度总线，决定某一时刻和哪个从设备通信。**同一时刻，I<sup>2</sup>C总线上只能有一对主设备、从设备通信。**
- 每个I<sup>2</sup>C从设备在I<sup>2</sup>C总线通讯中有一个I<sup>2</sup>C从设备地址，该地址唯一，是从设备的固有属性，通信中主设备通过从设备地址来找到从设备。



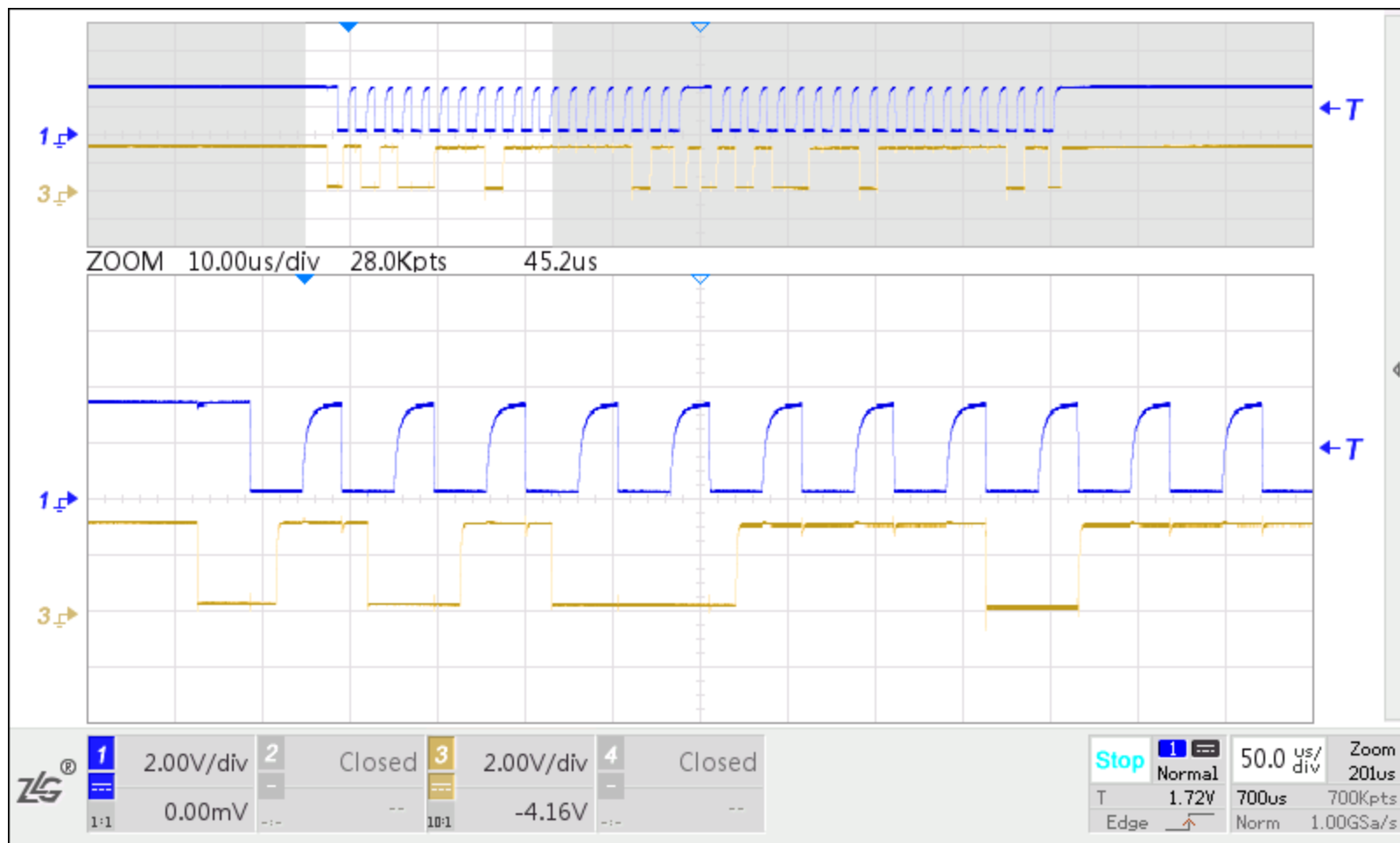


- I<sup>2</sup>C使用一个7bit的设备地址，一组总线最多和112个节点通信。连接到相同总线的I<sup>2</sup>C数量受到总线的最大电容400pF限制。
- 常见的I<sup>2</sup>C总线以传输速率的不同分为不同的模式：
  - 标准模式 (100Kbit/s)
  - 低速模式 (10Kbit/s)
  - 快速模式 (400Kbit/s)
  - 高速模式 (3.4Mbit/s)
- 时钟频率可以被下降到零，即暂停通信。





# 总线上电容过大会导致波形不完整



## • 通讯特征:

- 双向、二线制、串行、同步、非差分、低速率
  - **串行通信**，所有的数据以bit为单位在SDA线上串行传输
  - **同步通信**，即双方工作在同一个人时钟下，一般是通信的A方通过一根CLK信号线，将A设备的时钟传输到B设备，B设备在A设备传输的时钟下工作。**同步通信的特征是：通信线中有CLK。**
  - **非差分**，I<sup>2</sup>C通信速率不高，且通信距离近，使用电平信号通信。
  - **低速率**，I<sup>2</sup>C一般是同一个板子上的两个I<sup>2</sup>C芯片间通信，数据量不大，速率低。速率：几百KHz，速率可能不同，不能超过I<sup>2</sup>C的最高速率。



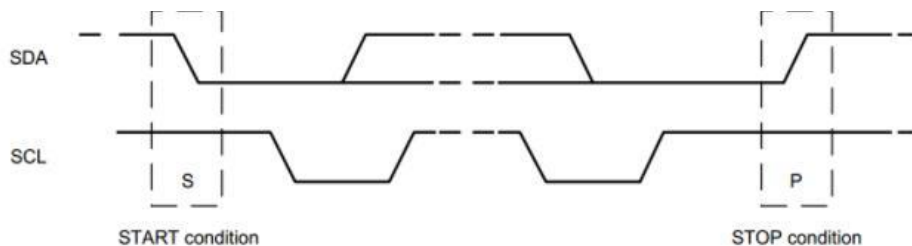
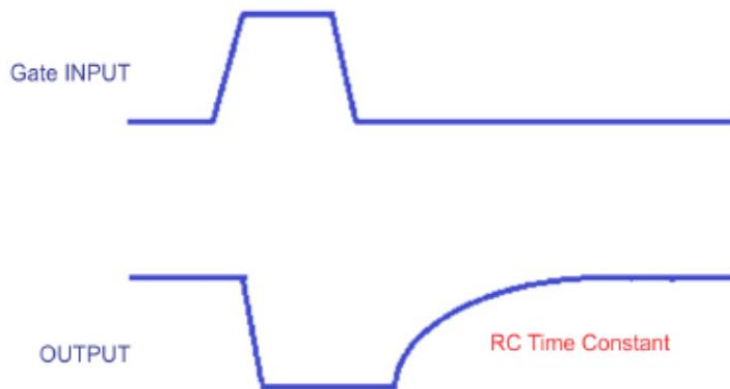
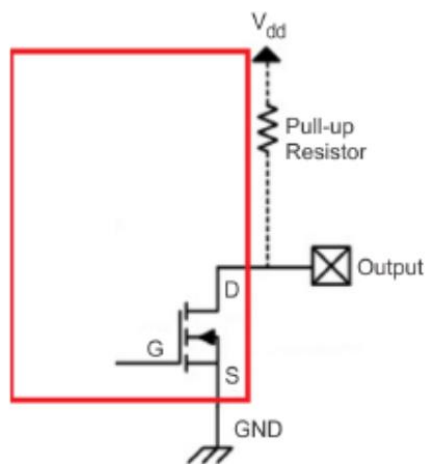
## ➤ I<sup>2</sup>C总线状态

- **空闲态**：没有设备发生通信。SDA和SCL都是高电平
- **忙态**：其中一个从设备和主设备通信，I<sup>2</sup>C总线被占用，其他从设备处于等待状态。



# ➤ I<sup>2</sup>C 总线通信协议

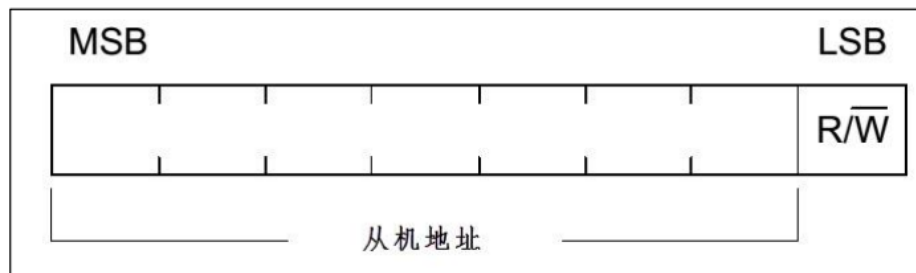
- I<sup>2</sup>C 总线通讯由**起始位**开始通讯，由**结束位**停止通讯，通讯结束释放 I<sup>2</sup>C 总线。起始位和结束位都由主设备发出。
- **起始位 (S)**：在 SCL 为高电平时，SDA 由**高电平**变为**低电平**
- **结束位 (P)**：在 SCL 为高电平时，SDA 由**低电平**变为**高电平**



START and STOP Conditions

## ➤ I<sup>2</sup>C 总线通信协议

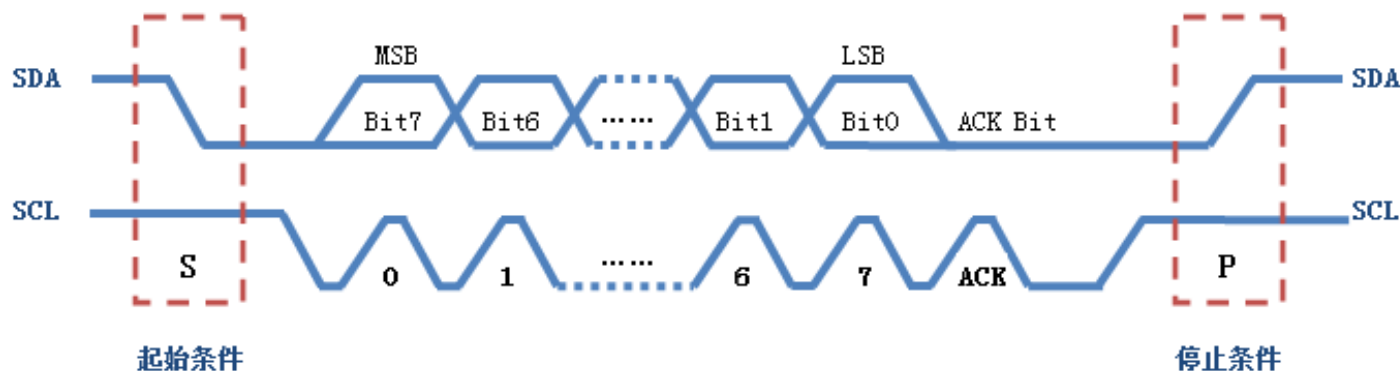
- 每个通信周期，主设备会先发**8位**的**从设备地址**（从设备地址由高**7位**的**实际从设备地址**和低**1位**的**读/写标志位**组成），
- 主设备以广播的形式发送从设备地址，I<sup>2</sup>C总线上的所有从设备收到地址后，判断从设备地址是否匹配，不匹配的从设备继续等待，**匹配的设备发出一个应答信号**。
- 同一时刻，主设备、从设备只能有一个设备发送数据。



## • 数据格式与应答

- I<sup>2</sup>C数据以**字节**（即8bits）为单位**传输**，每次可以传输的字节数量不受限制
- **每个字节传输完后**都会有一个**ACK应答**信号。应答信号的时钟是由主设备产生的。
- 先传输最高位(MSB)
- 如果从设备来不及处理主设备发送的数据，从设备会保持**SCL线为低电平**，强迫主设备等待从设备释放SCL线，直到**从设备处理完后，释放SCL线**，接着进行数据传输。
- 当**从机不能响应从机地址**时（例如它正在执行一些实时函数不能接收或发送），从机必须使数据线**SDA保持高电平**，主机然后产生一个停止条件终止传输或者产生重复起始条件开始新的传输。





I<sup>2</sup>C因为有两种寻址模式(7位寻址模式和10位寻址模式)，所以它具体的格式有两种，这里主要按分类部分来描述数据格式。总的数据传送为：主方(Master)每发送8个bit,必须等待从方(Slave)的一个反馈ACK。

S	从方地址+加读写控制	数据存放地址	数据	P
---	------------	--------	----	---

起始信号通常由主机发出，它作为一次传输的开始。在起始信号后总线被认为处于忙的状态

停止信号作为一次传送的结束，在该信号之后，总线被认为再次处于空闲状态。



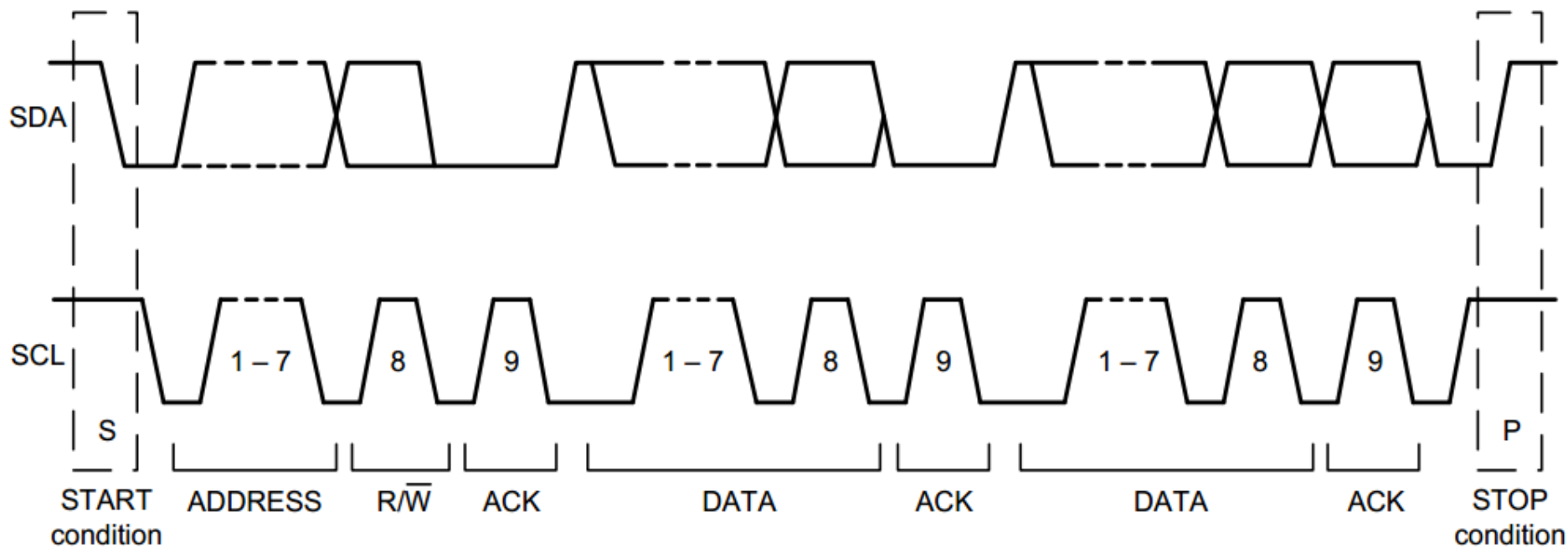
# ➤ 数据传输通讯

## - 写数据

- 先发送一个**起始位 (S)** ,
- 主设备发送一个地址数据 (由**7bit**的**从设备地址**+最低位的**写标志位**组成的**8bit**字节数据, 该读写标志位决定数据的传输方向)
- 主设备**释放SDA线**, 并等待从设备的应答信号 (ACK) 。
- **每个字节**数据的传输都要跟一个**应答信号位**。
- 数据传输以**停止位 (P)** 结束, 并且释放I2C总线。



## ➤ I<sup>2</sup>C 总线时序



**Complete I<sup>2</sup>C Data Transfer**



# • 数据传输通讯

## – 读数据

- 主设备发出一个起始位 (S)
- 主设备发送一个地址数据 (由7bit的从设备地址, 和最低位的写标志位组成的8bit字节数据, 该读写标志位决定数据的传输方向)
- 从设备回应(用来确定这个设备是否存在),
- 从设备传输数据, 传输数据之后, 要有一个回应信号 (确定数据是否接受完成)
- 传输下一个数据。
- 主设备发送一个停止信号。



## • 练习

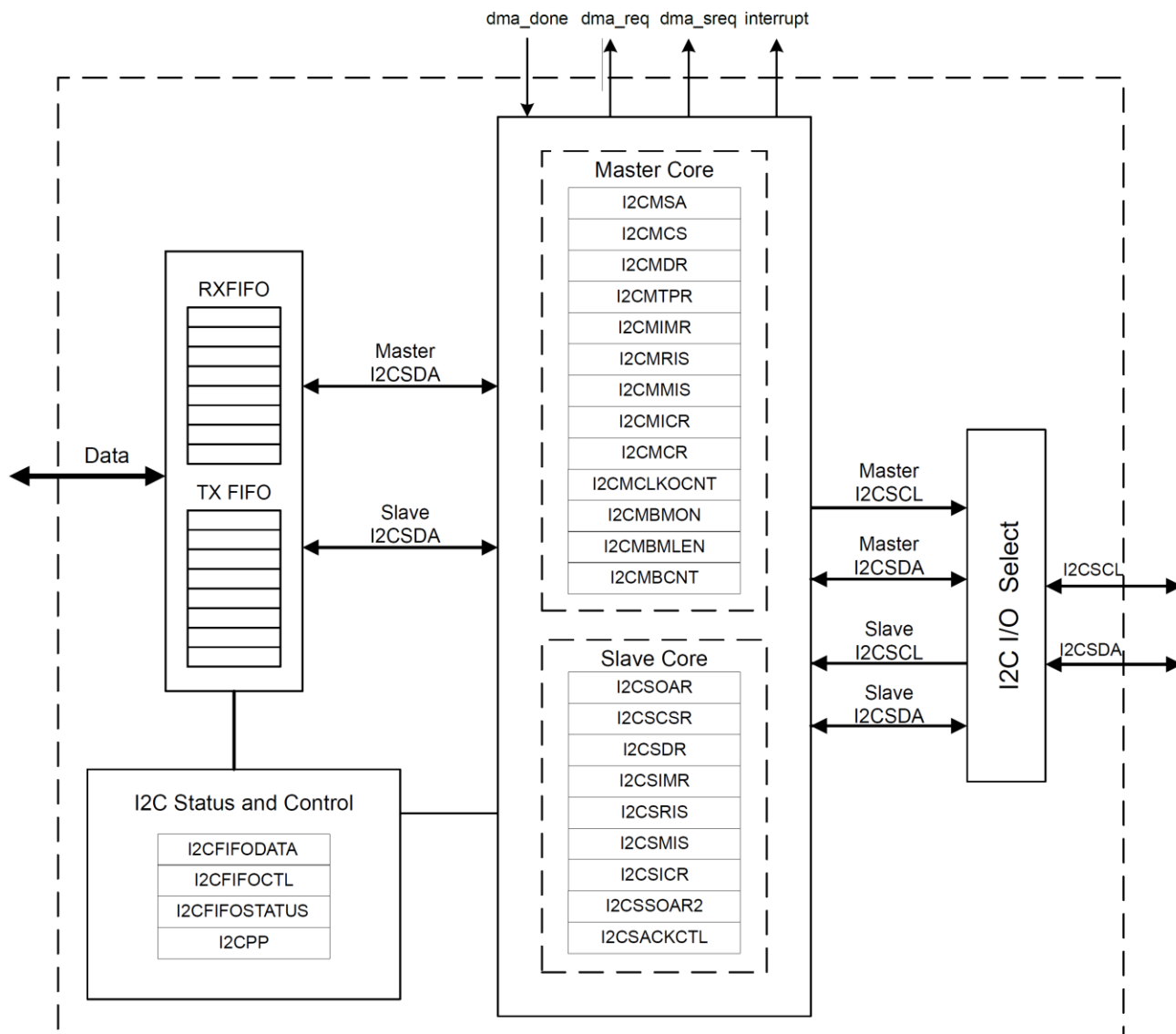
- 1. 向从机0x1D写一个数据0xAB， SCL和SDA的波形是样子，画出波形？标出那部分波形是主机发送的，哪部分是从机发送的。
- 2. 从从机0x1D读一个数据，如果这个数据是0xE5， SCL和SDA的波形是样子，画出波形？标出那部分波形是主机发送的，哪部分是从机发送的。



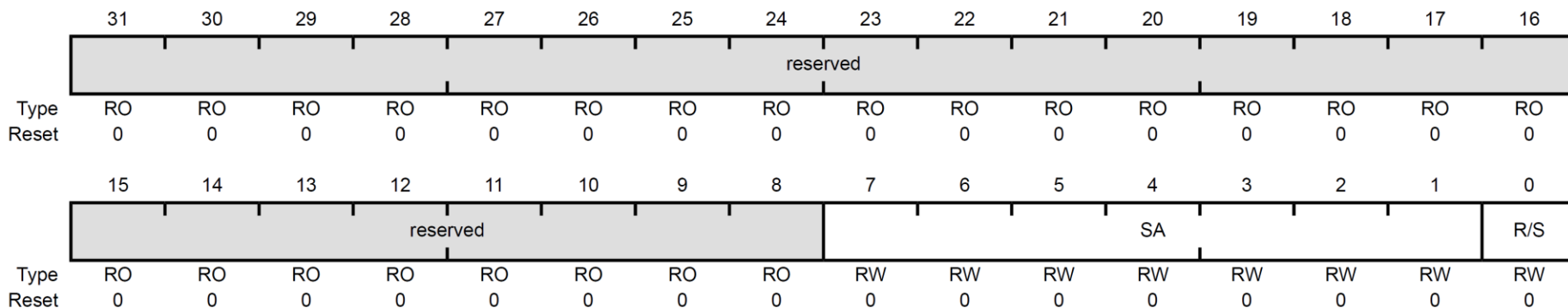
## ➤ TM4C的I<sup>2</sup>C模块

- 支持主/从收发模式
- 发送，接收都有FIFO
  - FIFO( First Input First Output)简单说就是指先进先出。在系统设计中，以增加数据传输率、处理大量数据流、匹配具有不同传输率的系统为目的，广泛使用FIFO存储器，从而提高了系统性能。FIFO存储器是一个先入先出的双口缓冲器，即第一个进入其内的数据第一个被移出。
- 支持四种传输速度
  - Standard (100 Kbps)
  - Fast-mode (400 Kbps)
  - Fast-mode plus (1 Mbps)
  - High-speed mode (3.33 Mbps)
- 支持发送接收中断
- 支持DMA





# ➤ I<sup>2</sup>C Master / Slave Address (I2CMSA)

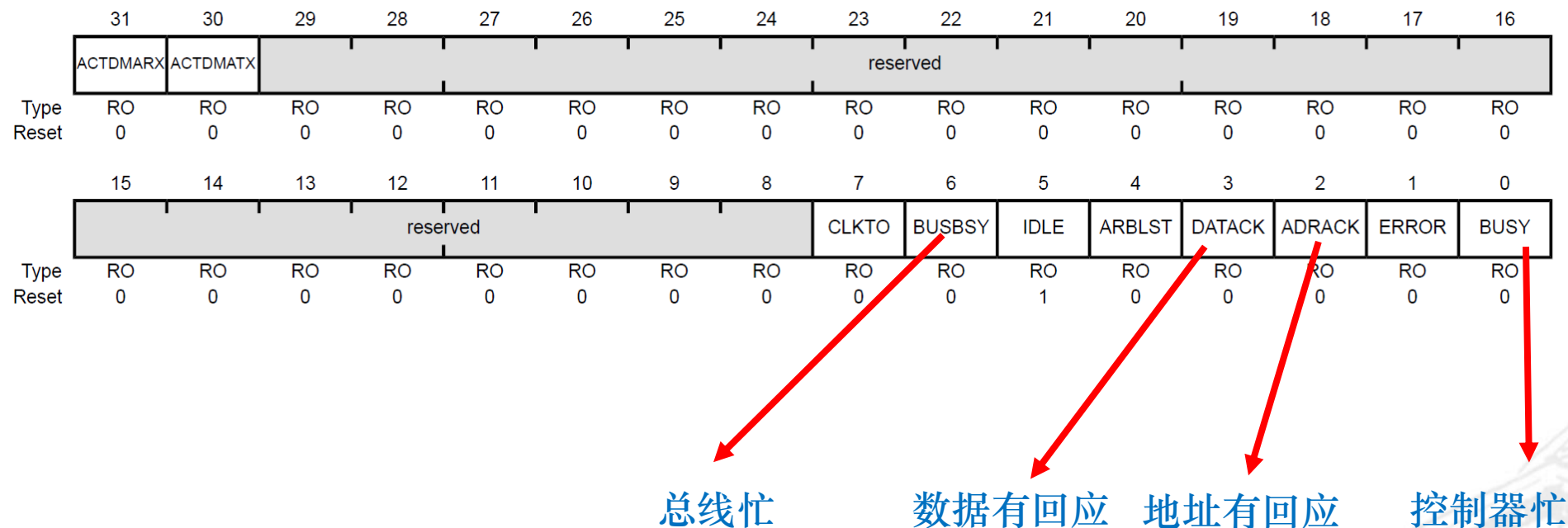


地址寄存器，最低位表示读/发送或写接收

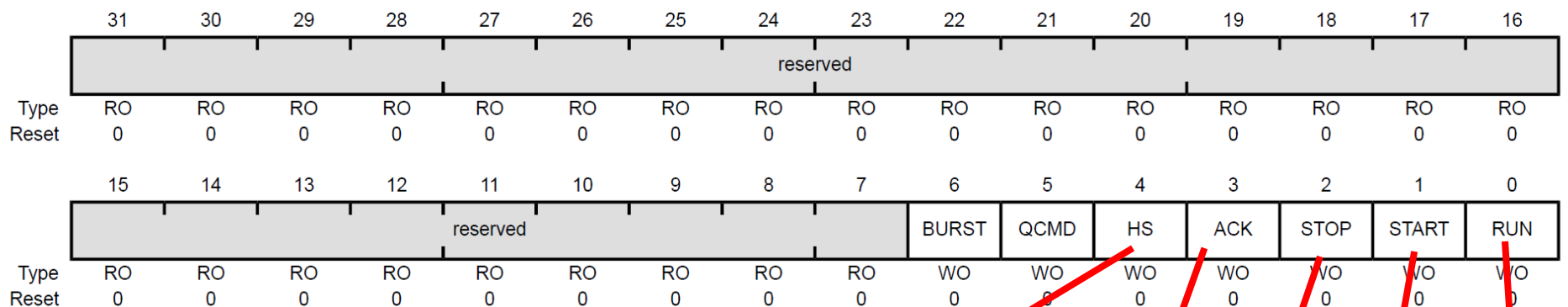




# ➤ I<sup>2</sup>C Master Control/Status (I<sup>2</sup>CMCS) 读



# • I<sup>2</sup>C Master Control/Status (I<sup>2</sup>CMCS) 写



高速模式

自动回应接收到的数据

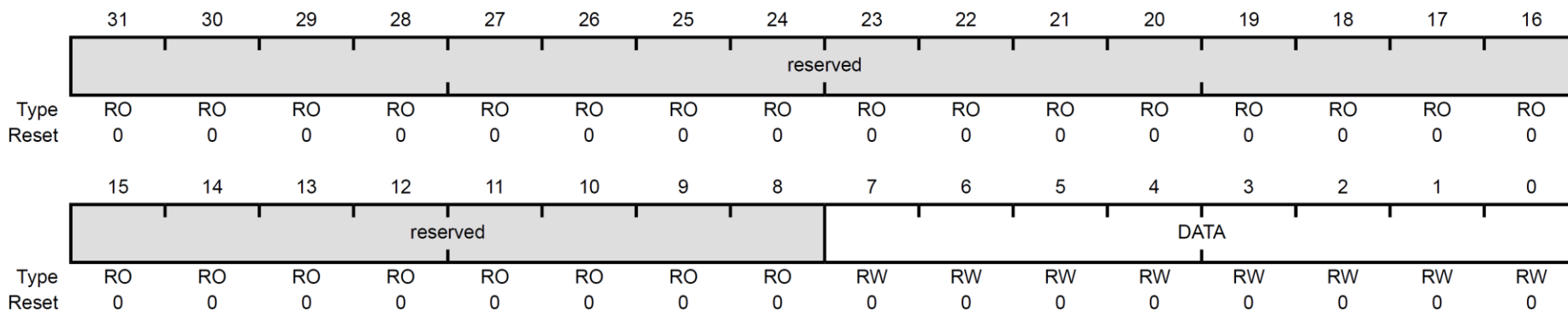
产生停止位

产生起始位

使能



# • I<sup>2</sup>C Master Data (I<sup>2</sup>CMDR)



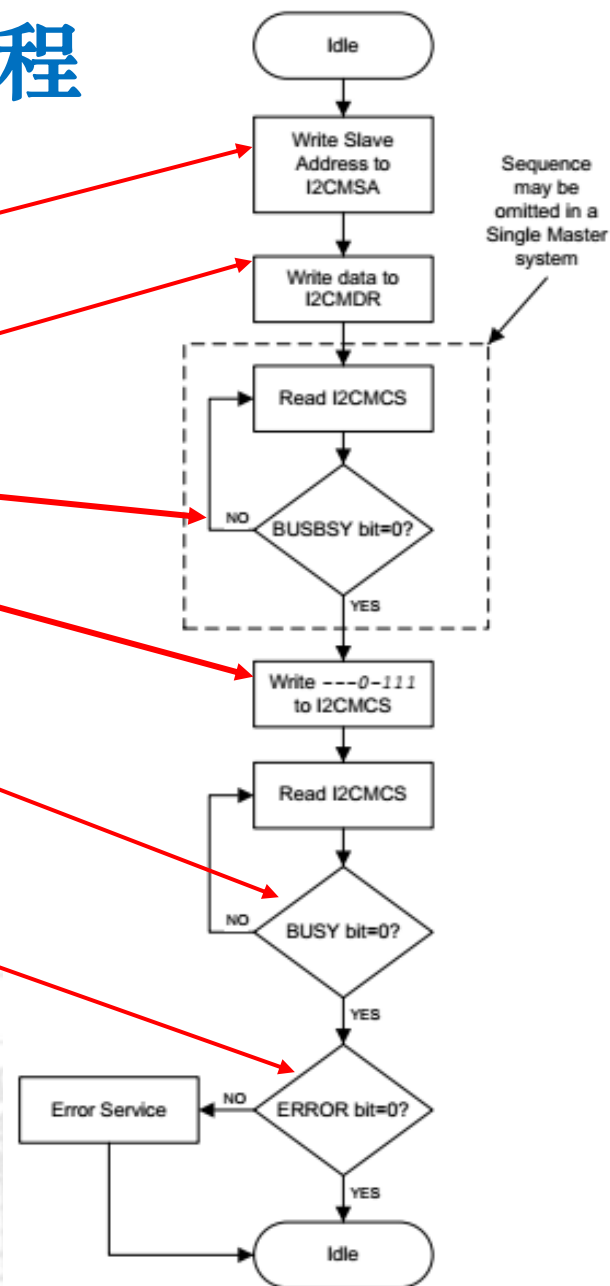
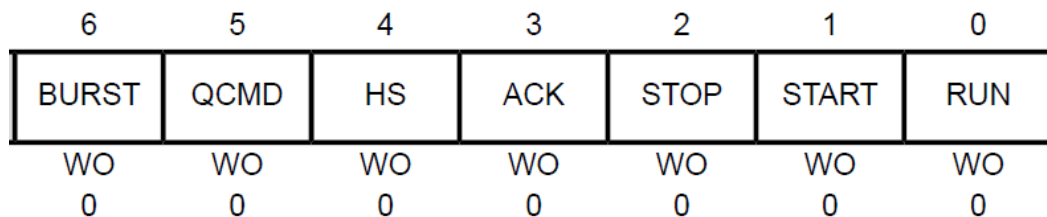
发送或接收的数据



# ➤ 发送和接收数据的具体操作流程

## - 1. 主设备写单个数据:

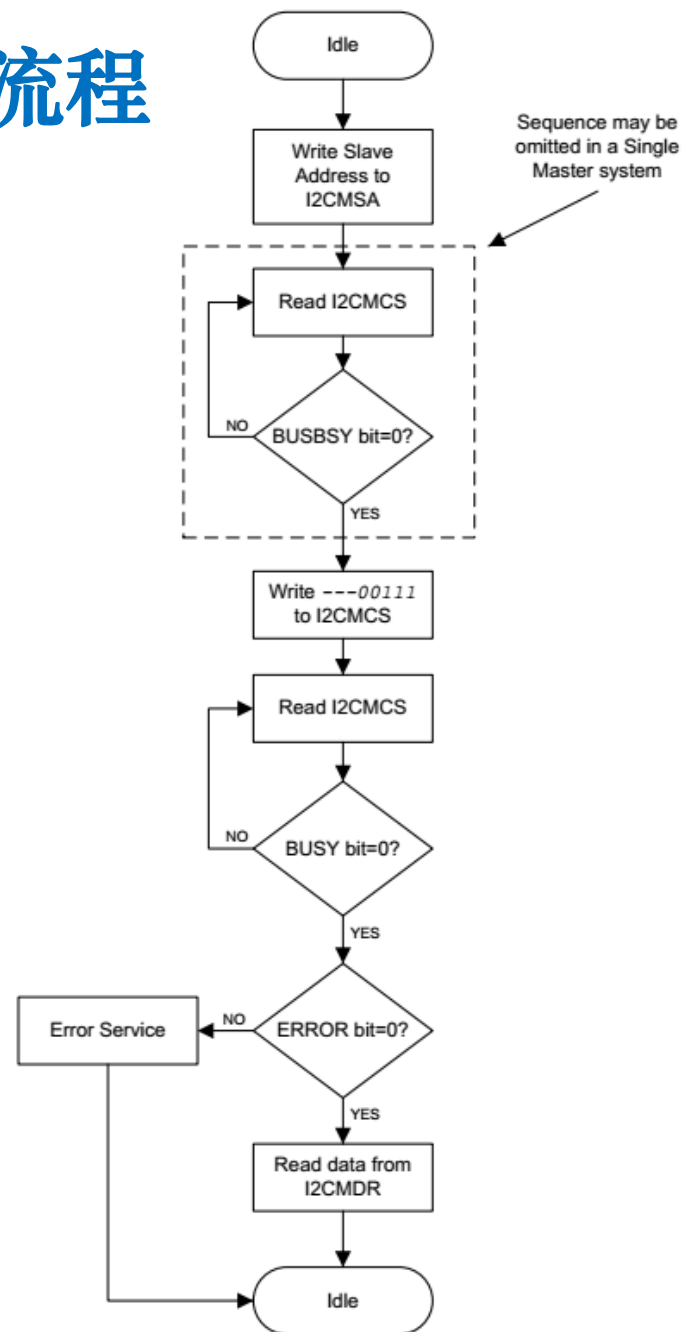
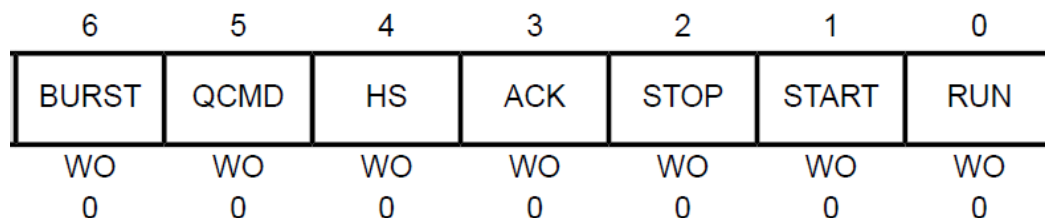
- 将从机地址写入I2CMSA
- 将要发送的数据写入 I2CMDR
- 检查总线是否忙碌
- 将---0-111写入I2CMCS
- 等到控制器忙碌结束
- 检查错误



# ➤ 发送和接收数据的具体操作流程

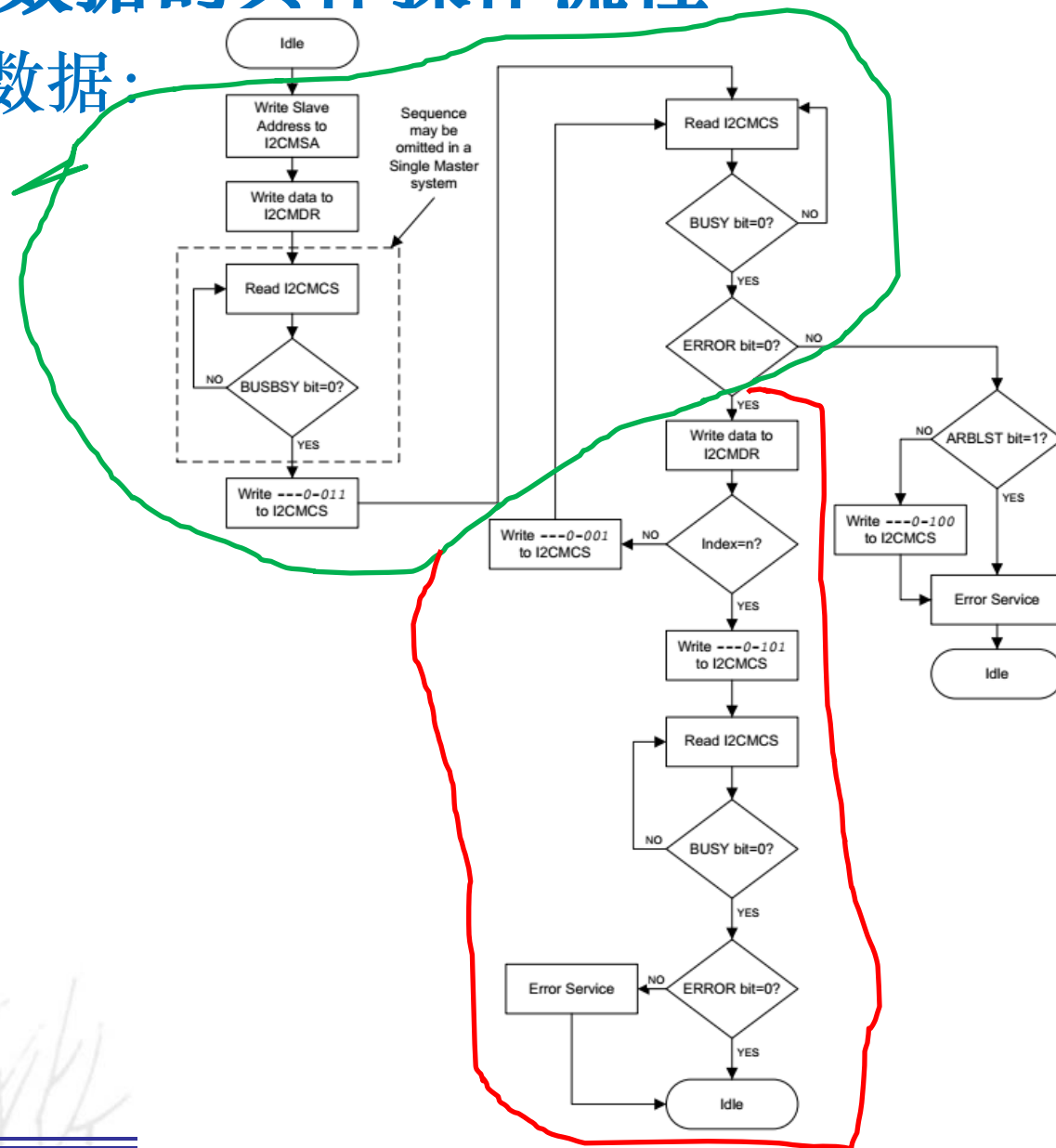
## - 2. 主设备读单个数据:

- 将从机地址写入I2CMSA
- 检查总线是否忙碌
- 将---00111写入I2CMCS
- 等到控制器忙碌结束
- 检查是否有错误
- 从I2CMDR读出数据



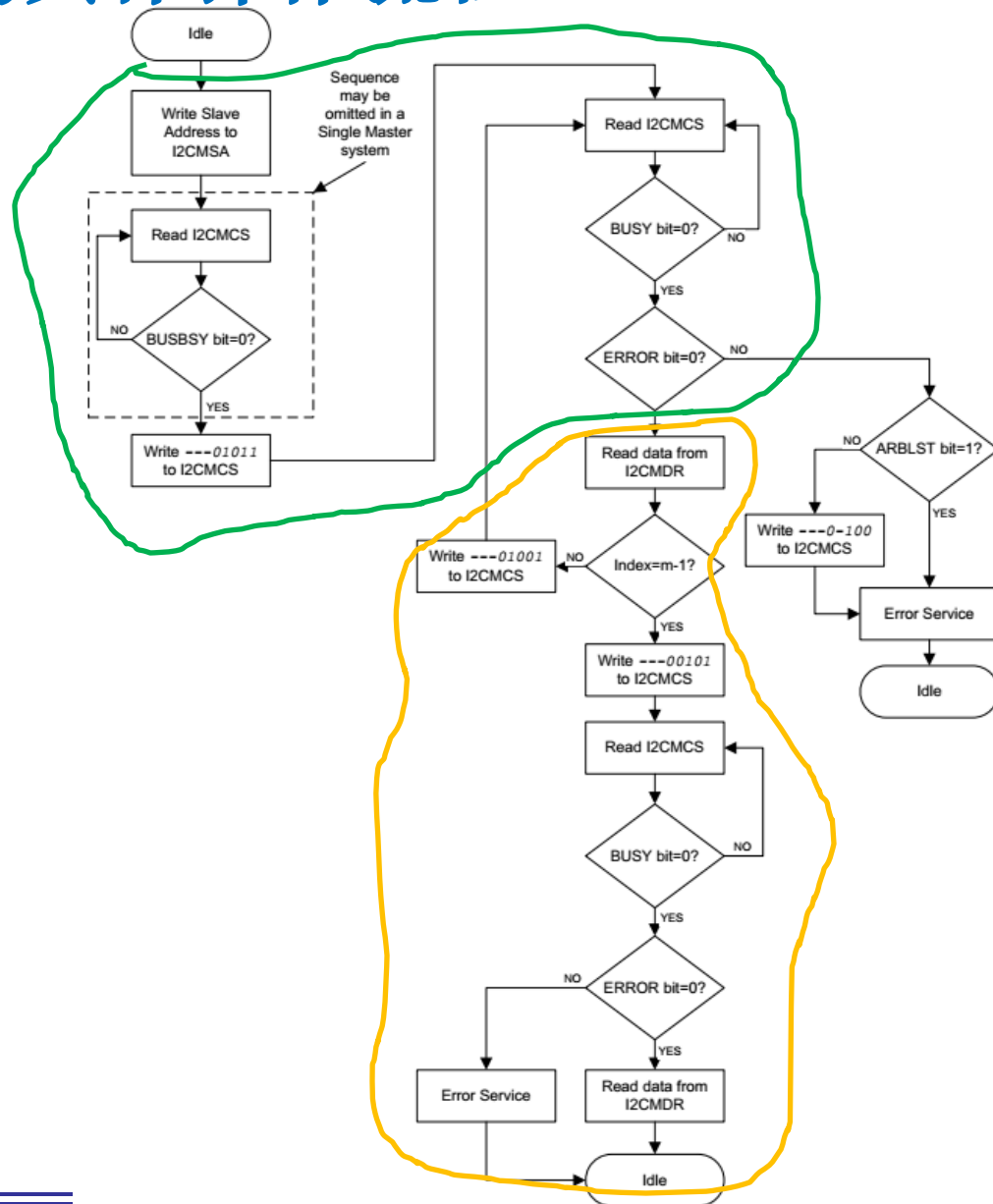
# ➤ 发送和接收数据的具体操作流程

## - 3. 写多个数据:



# ➤ 发送和接收数据的具体操作流程

## - 4. 读多个数据:





## ➤ 练习:

- 向从机0x1D写一个数据0x32, 应该怎样操作寄存器?
- 从从机0x1D读一个数据, 应该怎样操作寄存器?



## ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 1 在系统控制模块中，设置RCGCI2C寄存器，使能所需要用到的I<sup>2</sup>C模块。
- 2 在系统控制模块中，使能总线所在的GPIO模块的时钟。
- 3 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的复用功能。
- 4 设置I2CSDA引脚为漏极开路
- 5 配置GPIOCTL的PMCn位，将GPIO模块相应引脚的信号连接至I<sup>2</sup>C模块
- 6 向I2CMCR寄存器中写0x00000010初始化I2C模块
- 7 配置I2CMTPR寄存器，设置I<sup>2</sup>C模块的时钟
- 8 写I2CMSA寄存器，设置目标从机的地址，设置读写位
- 9 将要发送的数据写入I2CMDR寄存器
- 10 在I2CMCS寄存器中写入0x07 (STOP、START、RUN) ，发送数据。
- 11 查询I2CMCS的BUSBSY位，直到该位变为0。
- 12 检查错误。



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 1 在系统控制模块中，设置RCGCI2C寄存器，使能所需要用到的I<sup>2</sup>C模块（如I2C0）。

Inter-Integrated Circuit Run Mode Clock Gating Control (RCGCI2C)

Base 0x400F.E000

Offset 0x620

Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved						R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I<sup>2</sup>C模块时钟控制寄存器，控制10个I<sup>2</sup>C模块的时钟，R0至R9分别控制I<sup>2</sup>C0模块、I<sup>2</sup>C1模块至I<sup>2</sup>C9模块。



# • TM4C1294的I<sup>2</sup>C模块的使用

- 1 在系统控制模块中，设置**RCGCI2C**寄存器，**使能**所需要用到的**I<sup>2</sup>C模块**（如I<sup>2</sup>C0）。

方法1:

直接使用 **HWREGBITW(0x400FE000+0x620,0) = 1;**来操作该寄存器对应的位

方法2:

调用**TivaWare**库提供的函数: **SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);**

```
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
{
    ASSERT(_SysCtlPeripheralValid(ui32Peripheral));
    HWREGBITW(SYSCTL_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),
        ui32Peripheral & 0xff) = 1;
}
```

其中SCTL\_RCGCBASE的定义为:

```
#define SYSCTL_RCGCBASE    0x400fe600
```

输入参数SYSCTL\_PERIPH\_I2C0的定义为:

```
#define SYSCTL_PERIPH_I2C0    0xf0002000 // I2C 0
```

最后执行的代码为:

```
HWREGBITW(0x400FE620,0) = 1;
```

与直接操作寄存器的方式一致，但是可读性更强。



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 2 在系统控制模块中，使能总线所在的GPIO模块的时钟。
  - I2C总线的两根线，需要连接到实际引脚上，才能使用；
  - 在TM4C1294控制器中，I2C0模块的SCL连接到了PB2引脚上，I2C0模块的SDA连接到了PB3引脚上

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
I2C0SCL	91	PB2 (2)	I/O	OD	I <sup>2</sup> 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 <sup>2</sup> C module 0 data.

- 因此，如果用到了I2C0模块，需要使能GPIOB模块的时钟：  
`SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);`



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

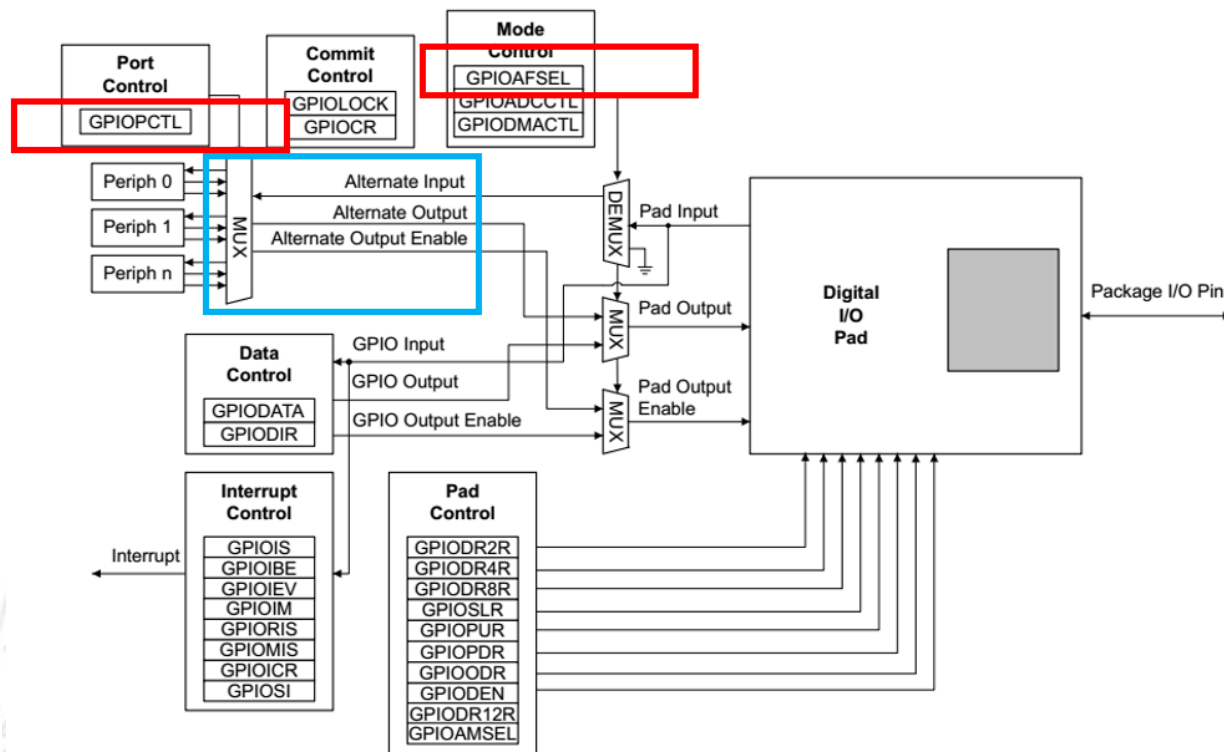
## ➤ 3 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的复用功能。

### 复用选择寄存器GPIOAFSEL

只有低八位有实际作用，用于控制该端口的八个引脚是否启用复用功能，每一位控制一个引脚。

0为不复用，该位对应的引脚为普通GPIO引脚

1为复用。复用的功能，由GPIOCTL决定



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## 3 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的复用功能。

### 复用选择寄存器GPIOAFSEL

只有低八位有实际作用，用于控制该端口的八个引脚是否启用复用功能，每一位控制一个引脚。

0为不复用，该位对应的引脚为普通GPIO引脚

1为复用。复用的功能，由GPIOCTL决定

#### GPIO Alternate Function Select (GPIOAFSEL)

GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (AHB) base: 0x4005.D000  
GPIO Port G (AHB) base: 0x4005.E000  
GPIO Port H (AHB) base: 0x4005.F000  
GPIO Port J (AHB) base: 0x4006.0000  
GPIO Port K (AHB) base: 0x4006.1000  
GPIO Port L (AHB) base: 0x4006.2000  
GPIO Port M (AHB) base: 0x4006.3000  
GPIO Port N (AHB) base: 0x4006.4000  
GPIO Port P (AHB) base: 0x4006.5000  
GPIO Port Q (AHB) base: 0x4006.6000  
Offset 0x420

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	AFSEL	RW	-	GPIO Alternate Function Select

#### Value Description

- |   |   |
|---|---|
| 0 | The associated pin functions as a GPIO and is controlled by the GPIO registers.                           |
| 1 | The associated pin functions as a peripheral signal and is controlled by the alternate hardware function. |
- The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 743.

$\text{HWREG}(0x40059000 + 0x420) = ((\text{HWREG}(0x40059000 + 0x420) | 0x0C));$





## ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 3 设置GPIO模块的**GPIOAFSEL**寄存器，配置**GPIO**的**复用功能**。
- 4 设置**I2C SDA**引脚为**漏极开路**。

调用GPIOPinTypeI2C和GPIOPinTypeI2CSCL函数：

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

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

```
void GPIOPinTypeI2C(uint32_t ui32Port, uint8_t ui8Pins)
```

```
{
```

```
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);
```

```
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,  
GPIO_PIN_TYPE_OD);
```

```
}
```

```
void
```

```
GPIOPinTypeI2CSCL(uint32_t ui32Port, uint8_t ui8Pins)
```

```
{
```

```
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);
```

```
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,  
GPIO_PIN_TYPE_STD);
```

```
}
```



## ➤ TM4C1294的I<sup>2</sup>C模块的使用

**void**

**GPIONDirModeSet**(uint32\_t ui32Port, uint8\_t ui8Pins, uint32\_t ui32PinIO)

{

// Check the arguments.

ASSERT(\_GPIOBaseValid(ui32Port));

ASSERT((ui32PinIO == GPIO\_DIR\_MODE\_IN) ||  
(ui32PinIO == GPIO\_DIR\_MODE\_OUT) ||  
(ui32PinIO == GPIO\_DIR\_MODE\_HW));

// Set the pin direction and mode.

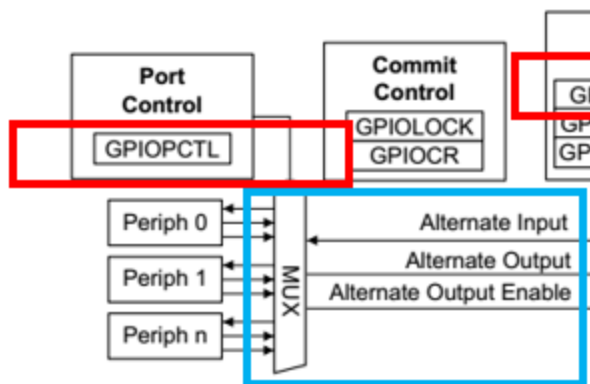
HWREG(ui32Port + GPIO\_O\_DIR) = ((ui32PinIO & 1) ?  
(HWREG(ui32Port + GPIO\_O\_DIR)|ui8Pins) :  
(HWREG(ui32Port + GPIO\_O\_DIR)& ~(ui8Pins)));  
HWREG(ui32Port + GPIO\_O\_AFSEL) = ((ui32PinIO & 2) ?  
(HWREG(ui32Port + GPIO\_O\_AFSEL)|ui8Pins) :  
(HWREG(ui32Port + GPIO\_O\_AFSEL)& ~(ui8Pins)));

}



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 5 配置GPIOCTL的PMCN位，将GPIO模块相应引脚的信号连接至I<sup>2</sup>C模块。



GPIOCTL寄存器中，由4位组成一个PMCN区域，控制一个引脚的复用功能。PMCN区域的值可设置为0-15(0000-1111)，分别代表不同的复用功能。

## GPIO Port Control (GPIOCTL)

GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (AHB) base: 0x4005.D000  
GPIO Port G (AHB) base: 0x4005.E000  
GPIO Port H (AHB) base: 0x4005.F000  
GPIO Port J (AHB) base: 0x4006.0000  
GPIO Port K (AHB) base: 0x4006.1000  
GPIO Port L (AHB) base: 0x4006.2000  
GPIO Port M (AHB) base: 0x4006.3000  
GPIO Port N (AHB) base: 0x4006.4000  
GPIO Port P (AHB) base: 0x4006.5000  
GPIO Port Q (AHB) base: 0x4006.6000  
Offset 0x52C

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PMC7				PMC6				PMC5				PMC4			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PMC3				PMC2				PMC1				PMC0			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 5 配置GPIOCTL的PMCN位，将GPIO模块相应引脚的信号连接至I<sup>2</sup>C模块。

Table 10-2. GPIO Pins and Alternate Functions (128TQFP) (continued)

IO	Pin	Analog or Special Function <sup>a</sup>	Digital Function (GPIOCTL PMCx Bit Field Encoding) <sup>b</sup>											
			1	2	3	4	5	6	7	8	11	13	14	15
PA5	38	-	U3Tx	I2C7SDA	T2CCP1	-	-	-	-	-	-	-	-	SSI0XDAT1
PA6	40	-	U2Rx	I2C6SCL	T3CCP0	-	USB0EPEN	-	-	-	-	SSI0XDAT2	-	EPI0S8
PA7	41	-	U2Tx	I2C6SDA	T3CCP1	-	USB0PFLT	-	-	-	USB0EPEN	SSI0XDAT3	-	EPI0S9
PB0	95	USB0ID	U1Rx	I2C5SCL	T4CCP0	-	-	-	CAN1Rx	-	-	-	-	-
PB1	96	USB0VBUS	U1Tx	I2C5SDA	T4CCP1	-	-	-	CAN1Tx	-	-	-	-	-
PB2	91	-	-	I2C0SCL	T5CCP0	-	-	-	-	-	-	-	USB0STP	EPI0S27
PB3	92	-	-	I2C0SDA	T5CCP1	-	-	-	-	-	-	-	USB0CLK	EPI0S28
PB4	121	AIN10	U0CTS	I2C5SCL	-	-	-	-	-	-	-	-	-	SSI1Fss
PB5	120	AIN11	U0RTS	I2C5SDA	-	-	-	-	-	-	-	-	-	SSI1Clk

如果将GPIOB的GPIOCTL寄存器的PMCN区域设置为2，则可以把PB2引脚与I2C0模块的SCL信号连接起来。  
如果将GPIOB的GPIOCTL寄存器的PMCN区域设置为2，则可以把PB3引脚与I2C0模块的SDA信号连接起来。



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 5 配置GPIOCTL的PMCn位，将GPIO模块相应引脚的信号连接至I<sup>2</sup>C模块。

方法1：直接操作寄存器来实现：

HWREG(GPIO\_PORTB\_BASE+ GPIO\_O\_PCTL)=0x2200;

方法2：用TivaWare库提供的函数GPIOPinConfigure来实现：

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL);

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA);

其中GPIO\_PB2\_I2C0SCL和GPIO\_PB3\_I2C0SDA这两个宏定义在pin\_map.h中：

#define GPIO\_PB2\_T5CCP0 0x00010803

#define GPIO\_PB2\_I2C0SCL 0x00010802

#define GPIO\_PB2\_USB0STP 0x0001080E

#define GPIO\_PB2\_EPI0S27 0x0001080F

#define GPIO\_PB3\_I2C0SDA 0x00010C02

#define GPIO\_PB3\_T5CCP1 0x00010C03

#define GPIO\_PB3\_USB0CLK 0x00010C0E


#define GPIO\_PB3\_EPI0S28 0x00010C0F



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 5 配置GPIOCTL的PMCn位，将GPIO模块相应引脚的信号连接至I<sup>2</sup>C模块。

```
void
GPIOPinConfigure(uint32_t ui32PinConfig)
{
    uint32_t ui32Base, ui32Shift;
    ASSERT(((ui32PinConfig >> 16) & 0xff) < 18);
    ASSERT(((ui32PinConfig >> 8) & 0xe3) == 0);
    ui32Base = (ui32PinConfig >> 16) & 0xff;
    if(HWREG(SYSCTL_GPIOHBCTL) & (1 << ui32Base))
    {
        ui32Base = g_pui32GPIOBaseAddrs[(ui32Base << 1) + 1];
    }
    else
    {
        ui32Base = g_pui32GPIOBaseAddrs[ui32Base << 1];
    }
    ui32Shift = (ui32PinConfig >> 8) & 0xff;
    HWREG(ui32Base + GPIO_O_PCTL) = ((HWREG(ui32Base + GPIO_O_PCTL) &
        ~(0xf << ui32Shift)) | ((ui32PinConfig & 0xf) << ui32Shift));
```

 #define GPIO\_PB2\_I2C0SCL 0x00010802  
#define GPIO\_PB3\_I2C0SDA 0x00010C02





# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 6 向I2CMCR寄存器中写0x00000010初始化I<sup>2</sup>C模块。

## I2C Master Configuration (I2CMCR)

I2C 0 base: 0x4002.0000

I2C 1 base: 0x4002.1000

I2C 2 base: 0x4002.2000

I2C 3 base: 0x4002.3000

I2C 4 base: 0x400C.0000

I2C 5 base: 0x400C.1000

I2C 6 base: 0x400C.2000

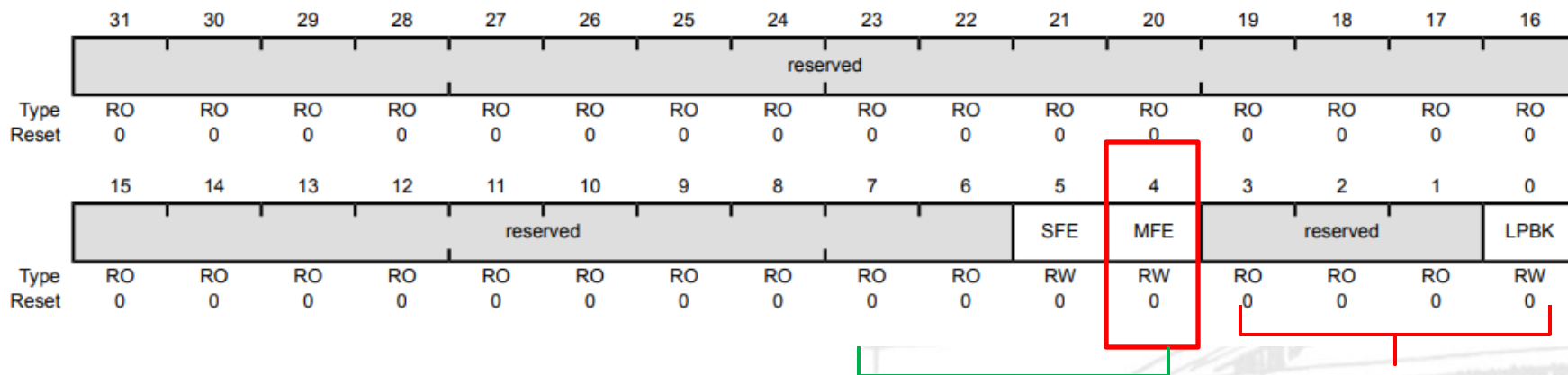
I2C 7 base: 0x400C.3000

I2C 8 base: 0x400B.8000

I2C 9 base: 0x400B.9000

Offset 0x020

Type RW, reset 0x0000.0000



直接操作寄存器，使能主站功能：

HWREG(I2C0\_BASE+ I2C\_O\_MCR) = 0x00000010;



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## - 7 配置I2CMTPR寄存器，设置I2C模块的时钟。

### I2C Master Timer Period (I2CMTPR)

I2C 0 base: 0x4002.0000

I2C 1 base: 0x4002.1000

I2C 2 base: 0x4002.2000

I2C 3 base: 0x4002.3000

I2C 4 base: 0x400C.0000

I2C 5 base: 0x400C.1000

I2C 6 base: 0x400C.2000

I2C 7 base: 0x400C.3000

I2C 8 base: 0x400B.8000

I2C 9 base: 0x400B.9000

Offset 0x00C

Type RW, reset 0x0000.0001

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved													PULSEL		
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								HS	TPR						
Type	RO	RO	RO	RO	RO	RO	RO	RO	WO	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1





# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## – 7 配置I2CMTPR寄存器，设置I<sup>2</sup>C模块的时钟。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved													PULSESEL		
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								HS	TPR						
Type	RO	RO	RO	RO	RO	RO	RO	RO	WO	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

TPR区域决定了I<sup>2</sup>C的时钟

$$TPR = (\text{System Clock} / (2 * (SCL\_LP + SCL\_HP) * SCL\_CLK)) - 1;$$

System Clock为系统时钟频率

SCL\_LP=6，SCL\_HP=4

SCL\_CLK为需要的I<sup>2</sup>C时钟

如果系统时钟为120MHz，使用100kHz的I<sup>2</sup>C时钟，那么TPR的值为：

$$TPR = (120\,000\,000 / (2 * (6 + 4) * 100\,000)) - 1 = 59$$

计算出TPR的值后，操作寄存器设置I2CMTPR寄存器：

$$HWREG(I2C0\_BASE + I2C\_O\_MTPR) = 59;$$



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## – 7 配置I2CMTPR寄存器，设置I<sup>2</sup>C模块的时钟。

TivaWare库提供了函数I2CmasterInitExpClk可以一次性初始化I<sup>2</sup>C模块，并设置时钟。

Initializes the I2C Master block.

### Prototype:

```
void  
I2CmasterInitExpClk (uint32_t ui32Base,  
                     uint32_t ui32I2CClk,  
                     bool bFast)
```

### Parameters:

**ui32Base** is the base address of the I2C module.

**ui32I2CClk** is the rate of the clock supplied to the I2C module.

**bFast** set up for fast data transfers.

### Description:

This function initializes operation of the I2C Master block by configuring the bus speed for the master and enabling the I2C Master block.

If the parameter *bFast* is **true**, then the master block is set up to transfer data at 400 Kbps; otherwise, it is set up to transfer data at 100 Kbps. If Fast Mode Plus (1 Mbps) is desired, software should manually write the I2CMTPR after calling this function. For High Speed (3.4 Mbps) mode, a specific command is used to switch to the faster clocks after the initial communication with the slave is done at either 100 Kbps or 400 Kbps.



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## – 7 配置I2CMTPR寄存器，设置I<sup>2</sup>C模块的时钟。

TivaWare库提供了函数I2CmasterInitExpClk可以一次性初始化I<sup>2</sup>C模块，并设置时钟。

```
I2CMasterInitExpClk(I2C0_BASE, g_ui32SysClock, false);
```

其中g\_ui32SysClock是系统时钟频率，由SysCtlClockFreqSet函数返回。  
以上代码将I2C0模块的传输速率设置为100kHz。



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

```
{
```

```
    uint32_t ui32SCLFreq;
```

```
    uint32_t ui32TPR;
```

```
    ASSERT(_I2CBaseValid(ui32Base));
```

```
    I2CMasterEnable(ui32Base);
```

```
    if(bFast == true)
```

```
    {
```

```
        ui32SCLFreq = 400000;
```

```
    }
```

```
    else
```

```
    {
```

```
        ui32SCLFreq = 100000;
```

```
    }
```

```
    ui32TPR = ((ui32I2CCLK + (2 * 10 * ui32SCLFreq) - 1) /  
              (2 * 10 * ui32SCLFreq)) - 1;
```

```
    HWREG(ui32Base + I2C_O_MTPR) = ui32TPR;
```

```
    if(HWREG(ui32Base + I2C_O_PP) & I2C_PP_HS)
```

```
    {
```

```
        ui32TPR = ((ui32I2CCLK + (2 * 3 * 3400000) - 1) /  
                  (2 * 3 * 3400000)) - 1;
```

```
        HWREG(ui32Base + I2C_O_MTPR) = I2C_MTPR_HS | ui32TPR;
```

```
    }
```

```
}
```

```
#define I2C_MCR_MFE
```

```
0x00000010
```

```
I2CMasterEnable(uint32_t ui32Base)
```

```
{
```

```
    ASSERT(_I2CBaseValid(ui32Base));
```

```
    HWREG(ui32Base + I2C_O_MCR) |= I2C_MCR_MFE;
```

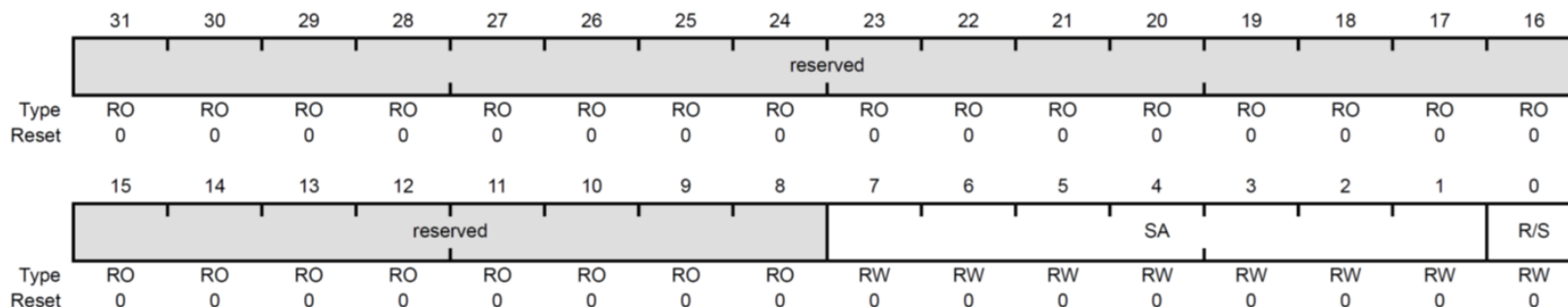
```
}
```



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 8 写I2CMSA寄存器，设置目标从机的地址，设置读写位。

## I2C Master Slave Address (I2CMSA)



方法1:

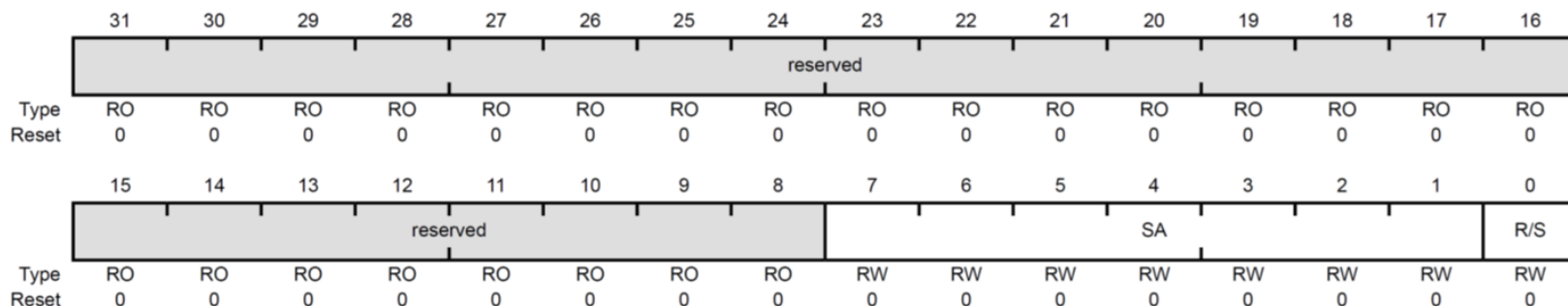
直接操作寄存器设置I2CMSA寄存器:  $\text{HWREG}(\text{I2C0\_BASE} + \text{I2C\_O\_MSA}) = \text{????};$



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 8 写I2CMSA寄存器，设置目标从机的地址，设置读写位。

## I2C Master Slave Address (I2CMSA)



方法2：使用TivaWare库函数I2CMasterSlaveAddrSet设置I<sup>2</sup>C MSA寄存器。

void

I2CMasterSlaveAddrSet(uint32\_t ui32Base, uint8\_t ui8SlaveAddr, bool bReceive)

{

ASSERT(\_I2CBaseValid(ui32Base));

ASSERT(!(ui8SlaveAddr & 0x80));

HWREG(ui32Base + I2C\_O\_MSA) = (ui8SlaveAddr << 1) | bReceive;

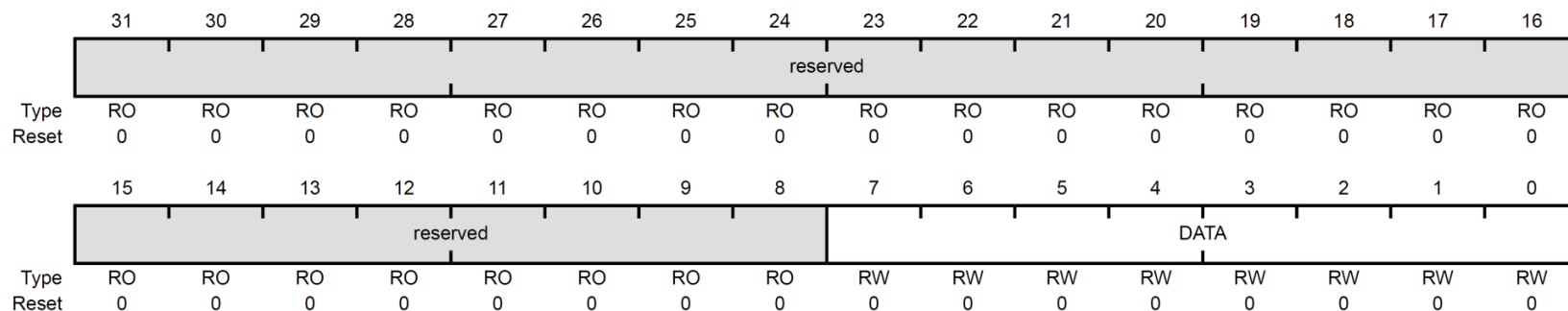
}



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## – 9 把要发送的数据写入I2CMDR寄存器。

I2C Master Data (I2CMDR)



调用TivaWare库的I2CMasterDataPut函数

```
void  
I2CMasterDataPut(uint32_t ui32Base, uint8_t ui8Data)  
{  
    ASSERT(_I2CBaseValid(ui32Base));  
    HWREG(ui32Base + I2C_O_MDR) = ui8Data;  
}
```



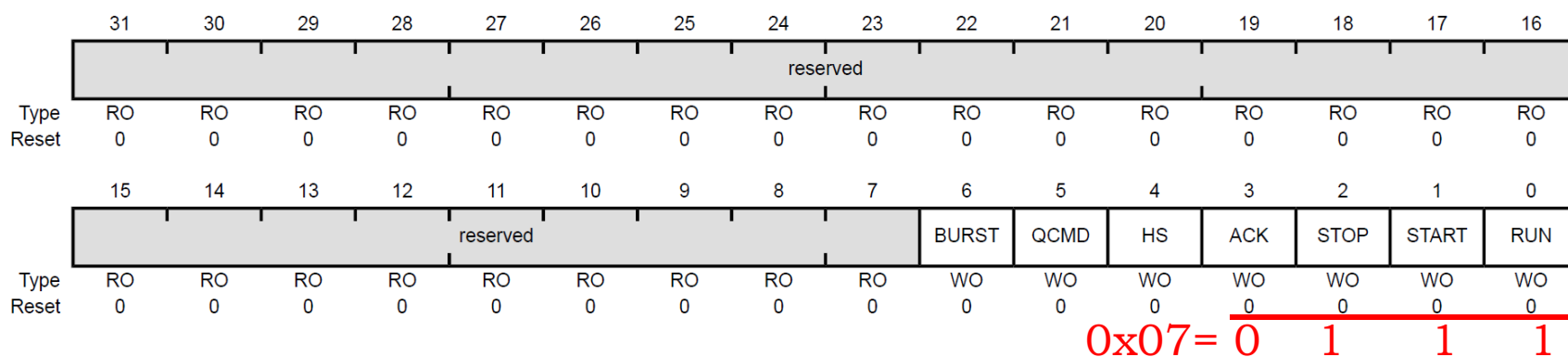


# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 10 在I2CMCS寄存器中写入0x07 (STOP、START、RUN) , 发送数据。

I<sup>2</sup>C Master Control/Status (I2CMCS)

写I2CMCS:



BURST: 突发模式

HS: 高速模式

ACK: 自动回应接收到的数据

STOP: 产生停止位

START: 产生起始位

RUN: 开始传输



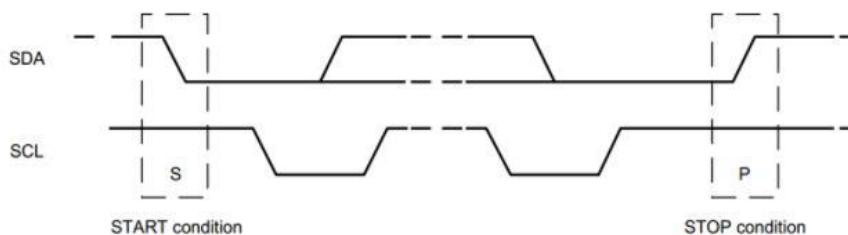
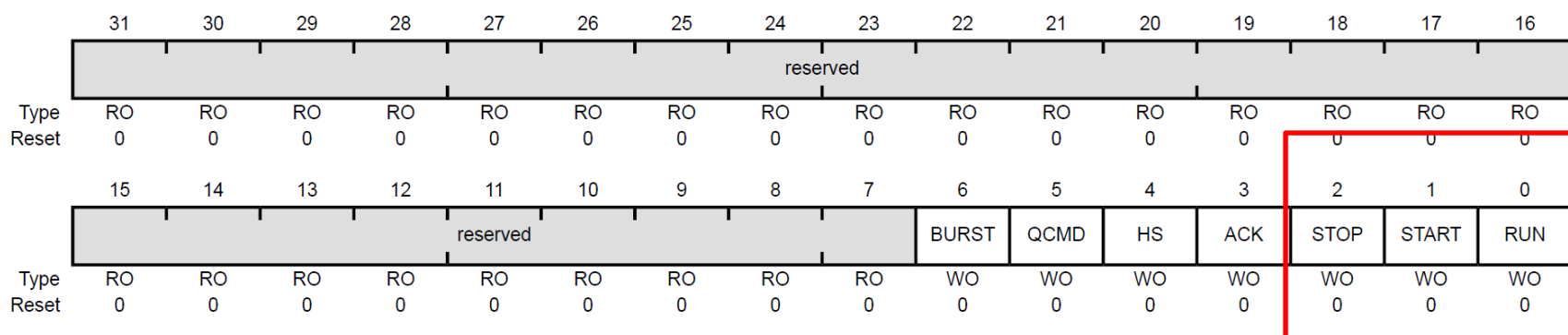


# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 10 在I2CMCS寄存器中写入0x07 (STOP、START、RUN) , 发送数据。

I<sup>2</sup>C Master Control/Status (I2CMCS)

写I2CMCS:



START and STOP Conditions

起始条件	从站地址 (7bits)	写 (1bit)	ACK(1bit)	数据 (8bits)	结束条件
------	--------------	----------	-----------	------------	------



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 10 在I2CMCS寄存器中写入0x07 (STOP、START、RUN) , 发送数据。

I<sup>2</sup>C Master Control/Status (I2CMCS)

写I2CMCS:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								BURST		QCMD	HS	ACK	STOP	START	RUN
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	WO	WO	WO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TivaWare库提供了I2CMasterControl函数, 设置I2CMCS寄存器

I2CMasterControl(uint32\_t ui32Base, uint32\_t ui32Cmd)



void

**I2CMasterControl**(uint32\_t ui32Base, uint32\_t **ui32Cmd**)

```
{  
    ASSERT(!_I2CBaseValid(ui32Base));  
    ASSERT((ui32Cmd == I2C_MASTER_CMD_SINGLE_SEND) ||  
           (ui32Cmd == I2C_MASTER_CMD_SINGLE_RECEIVE) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_SEND_START) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_SEND_CONT) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_SEND_FINISH) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_SEND_ERROR_STOP) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_RECEIVE_START) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_RECEIVE_CONT) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_RECEIVE_FINISH) ||  
           (ui32Cmd == I2C_MASTER_CMD_BURST_RECEIVE_ERROR_STOP) ||  
           (ui32Cmd == I2C_MASTER_CMD_QUICK_COMMAND) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_SINGLE_SEND) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_SEND_START) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_SEND_CONT) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_SEND_FINISH) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_SEND_ERROR_STOP) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_RECEIVE_START) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_RECEIVE_CONT) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_RECEIVE_FINISH) ||  
           (ui32Cmd == I2C_MASTER_CMD_FIFO_BURST_RECEIVE_ERROR_STOP) ||  
           (ui32Cmd == I2C_MASTER_CMD_HS_MASTER_CODE_SEND));
```

HWREG(ui32Base + **I2C\_O\_MCS**) = **ui32Cmd**;

}



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

- 11 查询I2CMCS的**BUSBSY**位，直到该位变为0。

I<sup>2</sup>C Master Control/Status (**I2CMCS**)

读**I2CMCS**:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ACTDMARX	ACTDMATX	reserved													
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								CLKTO	BUSBSY	IDLE	ARBLST	DATAACK	ADRACK	ERROR	BUSY
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

**bool I2CMasterBusBusy(uint32\_t ui32Base)**

```
{  
    ASSERT(_I2CBaseValid(ui32Base));  
    if(HWREG(ui32Base + I2C_O_MCS) & I2C_MCS_BUSBSY)  
    {  
        return(true);  
    } else  
    {  
        return(false);  
    }  
}
```

#define I2C\_MCS\_BUSBSY 0x00000040 // Bus Busy



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

## – 12 检查错误。

I<sup>2</sup>C Master Control/Status (I2CMCS)

读I2CMCS:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ACTDMARX	ACTDMATX	reserved													
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								CLKTO	BUSBSY	IDLE	ARBLST	DATAACK	ADRACK	ERROR	BUSY
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

调用TivaWare库函数I2CMasterErr

如果I2C控制器已经发送完成，且发生了错误或者仲裁失败，则返回错误信息。



---

uint32\_t

**I2CMasterErr**(uint32\_t ui32Base)

```
{
    uint32_t ui32Err;
    ASSERT(!_I2CBaseValid(ui32Base));
    ui32Err = HWREG(ui32Base + I2C_O_MCS);
    if(ui32Err & I2C_MCS_BUSY)
    {
        return(I2C_MASTER_ERR_NONE);
    }
    if(ui32Err & (I2C_MCS_ERROR | I2C_MCS_ARBLST))
    {
        return(ui32Err & (I2C_MCS_ARBLST | I2C_MCS_DATAACK | I2C_MCS_ADRACK));
    }
    else
    {
        return(I2C_MASTER_ERR_NONE);
    }
}
```



# ➤ TM4C1294的I<sup>2</sup>C模块的使用

– 示例:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);  
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);  
GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
```

```
GPIOPinConfigure(GPIO_PB2_I2C0SCL);  
GPIOPinConfigure(GPIO_PB3_I2C0SDA);
```

```
I2CMasterInitExpClk(I2C0_BASE, g_ui32SysClock, false);
```

```
while(I2CMasterBusBusy(I2C0_BASE));  
while(I2CMasterBusy(I2C0_BASE)); 从机地址 0  
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);  
I2CMasterDataPut(I2C0_BASE, data);  
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND); //0x07  
do{SysCtlDelay(400);} while(I2CMasterBusBusy(I2C0_BASE));  
I2CState=I2CMasterErr(I2C0_BASE);
```



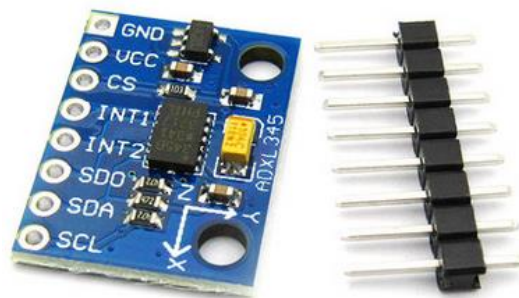


## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

能够在预先不知道物体运动方向的情况下，准确且全面的测量出物体的空间加速度，并且体积小

- (1) 汽车电子：
- (2) 便携式设备的抗冲击防护：
- (3) 卫星导航：
- (4) 虚拟现实：



ADXL345使用I2C通信，只需要连接4根线：

- VCC：电源线3.3V-5V
- GND：地线
- SDA：I2C总线的数据线，上拉
- SCL：I2C总线的时钟线，上拉

表5. 引脚功能描述

引脚编号	引脚名称	描述
1	V <sub>DDIO</sub>	数字接口电源电压。
2	GND	该引脚必须接地。
3	RESERVED	保留。该引脚必须连接到V <sub>s</sub> 或保持断开。
4	GND	该引脚必须接地。
5	GND	该引脚必须接地。
6	V <sub>s</sub>	电源电压。
7	$\overline{CS}$	片选。
8	INT1	中断1输出。
9	INT2	中断2输出。
10	NC	内部不连接。
11	RESERVED	保留。该引脚必须接地或保持断开。
12	SDO/ALT ADDRESS	串行数据输出(SPI 4线)/备用PC地址选择(I2C)
13	SDA/SDI/SDIO	串行数据(I <sup>2</sup> C)/串行数据输入(SPI 4线)/串行数据输入和输出(SPI 3线)。
14	SCL/SCLK	串行通信时钟。SCL为I <sup>2</sup> C时钟，SCLK为SPI时钟。





## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

ADXL345使用I<sup>2</sup>C通信，只需要连接4根线:

- VCC: 电源线3.3V-5V
- GND: 地线
- SDA: I<sup>2</sup>C总线的数据线, 上拉
- SCL: I<sup>2</sup>C总线的时钟线, 上拉

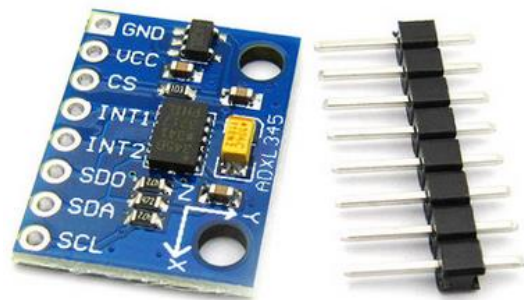
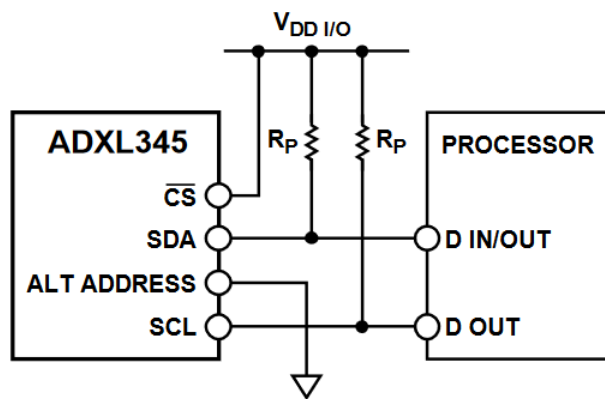


表5. 引脚功能描述

引脚编号	引脚名称	描述
1	V <sub>DD I/O</sub>	数字接口电源电压。
2	GND	该引脚必须接地。
3	RESERVED	保留。该引脚必须连接到V <sub>s</sub> 或保持断开。
4	GND	该引脚必须接地。
5	GND	该引脚必须接地。
6	V <sub>s</sub>	电源电压。
7	$\overline{CS}$	片选。
8	INT1	中断1输出。
9	INT2	中断2输出。
10	NC	内部不连接。
11	RESERVED	保留。该引脚必须接地或保持断开。
12	SDO/ALT ADDRESS	串行数据输出(SPI 4线)/备用I <sup>2</sup> C地址选择(I <sup>2</sup> C)
13	SDA/SDI/SDIO	串行数据(I <sup>2</sup> C)/串行数据输入(SPI 4线)/串行数据输入和输出(SPI 3线)。
14	SCL/SCLK	串行通信时钟。SCL为I <sup>2</sup> C时钟, SCLK为SPI时钟。



07925-008

将CS引脚拉高, ADXL345处于I<sup>2</sup>C模式  
支持标准(100 kHz)和快速(400 kHz)数  
据传输模式

ALT ADDRESS引脚接高电平, 器件的地址  
是0x1D,  
ALT ADDRESS引脚接地, 器件的地址是  
0x53(随后为R / W位)。



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据:

地址		名称	类型	复位值	描述
十六进制	十进制				
0x00	0	DEVID	R	11100101	器件ID
0x2C	44	BW_RATE	R/ $\overline{W}$	00001010	数据速率及功率模式控制
0x2D	45	POWER_CTL	R/ $\overline{W}$	00000000	省电特性控制
0x2E	46	INT_ENABLE	R/ $\overline{W}$	00000000	中断使能控制
0x2F	47	INT_MAP	R/ $\overline{W}$	00000000	中断映射控制
0x30	48	INT_SOURCE	R	00000010	中断源
0x31	49	DATA_FORMAT	R/ $\overline{W}$	00000000	数据格式控制
0x32	50	DATA_X0	R	00000000	X轴数据0
0x33	51	DATA_X1	R	00000000	X轴数据1
0x34	52	DATA_Y0	R	00000000	Y轴数据0
0x35	53	DATA_Y1	R	00000000	Y轴数据1
0x36	54	DATA_Z0	R	00000000	Z轴数据0
0x37	55	DATA_Z1	R	00000000	Z轴数据1
0x38	56	FIFO_CTL	R/ $\overline{W}$	00000000	FIFO控制
0x39	57	FIFO_STATUS	R	00000000	FIFO状态



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据：

#### 1. DEVID 器件编号

此寄存器里的数据位固定值**0b11100101**，如果从这个寄存器里**读**出的数据(即**0b11100101**)**正确**，说明I<sup>2</sup>C通信能够**正常工作**。



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据：

### 2. DATA\_FORMAT数据格式寄存器

#### 寄存器0x31—DATA\_FORMAT(读/写)

D7	D6	D5	D4	D3	D2	D1	D0
自测	SPI	INT_INVERT	0	FULL_RES	对齐	范围	

DATA\_FORMAT寄存器通过寄存器0x37控制寄存器0x32的数据显示。除±16 g范围以外的所有数据必须剪除，避免翻覆。

#### SELF\_TEST位

SELF\_TEST位设置为1，自测力应用至传感器，造成输出数据转换。值为0时，禁用自测力。

#### SPI位

SPI位值为1，设置器件为3线式SPI模式，值为0，则设置为4线式SPI模式。

#### INT\_INVERT位

INT\_INVERT位值为0，设置中断至高电平有效，值为1，则设置至低电平有效。

#### FULL\_RES位

当此位值设置为1，该器件为全分辨率模式，输出分辨率随着范围位设置的g范围，以4 mg/LSB的比例因子而增加。FULL\_RES位设置为0时，该器件为10位模式，范围位决定最大g范围和比例因子。

#### 对齐位

对齐位设置为1，选择左对齐(MSB)模式，设置为0，选择右对齐模式，并带有符号扩展功能。

#### 范围位

这些位设置g范围，如表21所述。

表21. g范围设置

设置		g范围
D1	D0	
0	0	±2 g
0	1	±4 g
1	0	±8 g
1	1	±16 g



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据:

### 3. 省电特性控制POWER\_CTL

寄存器0x2D—POWER\_CTL(读/写)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	链接	AUTO_SLEEP	测量	休眠	唤醒	

### 测量位

测量位设置为0，将器件置于待机模式，设置为1，置于测量模式。ADXL345待机模式下，以最小功耗上电。





## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据:

**4. 输出加速度数据: DATAx0、DATAx1、DATAY0、DATAY1、DATAZ0和DATAZ1**

**寄存器0x32至0x37—DATAx0、DATAx1、DATAY0、DATAY1、DATAZ0和DATAZ1(只读)**

这6个字节(寄存器0x32至寄存器0x37)都为8位字节, 保存各轴的输出数据。寄存器0x32和0x33保存x轴输出数据, 寄存器0x34和0x35保存y轴输出数据, 寄存器0x36和0x37保存z轴输出数据。输出数据为二进制补码, DATAx0为最低有效字节, DATAx1为最高有效字节, 其中x代表X、Y或Z。DATA\_FORMAT寄存器(地址0x31)控制数据格式。建议所有寄存器执行多字节读取, 以防止相继寄存器读取之间的数据变化。



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- ADXL345的通过内部寄存器控制芯片并读写数据：

1. 读取DEVID 器件编号，确认I2C总线正常工作 读寄存器
2. 设置测量范围（写DATA\_FORMAT寄存器，只设置一次） 写寄存器
3. 进入测量模式（写POWER\_CTL寄存器，只设置一次） 写寄存器
4. 读取数据（DATA0、DATA1、DATA0、DATA1、DATA0和DATA1寄存器） 读多个寄存器



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

读单个寄存器:

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

SINGLE-BYTE READ									
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START <sup>†</sup>	SLAVE ADDRESS + READ		
SLAVE			ACK		ACK			ACK	DATA

```
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
while(I2CMasterBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);
I2CMasterDataPut(I2C0_BASE, reg);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START); //0x03
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CRead);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE); //0x07
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CState=I2CMasterErr(I2C0_BASE);
return I2CMasterDataGet(I2C0_BASE);
```





## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

写单个寄存器（方法一）：

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

SINGLE-BYTE WRITE								
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		STOP
SLAVE			ACK		ACK		ACK	

```
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
while(I2CMasterBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);
I2CMasterDataPut(I2C0_BASE, reg);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START); //0x03
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);
I2CMasterDataPut(I2C0_BASE, value);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH); //0x05
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CState=I2CMasterErr(I2C0_BASE);
```



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

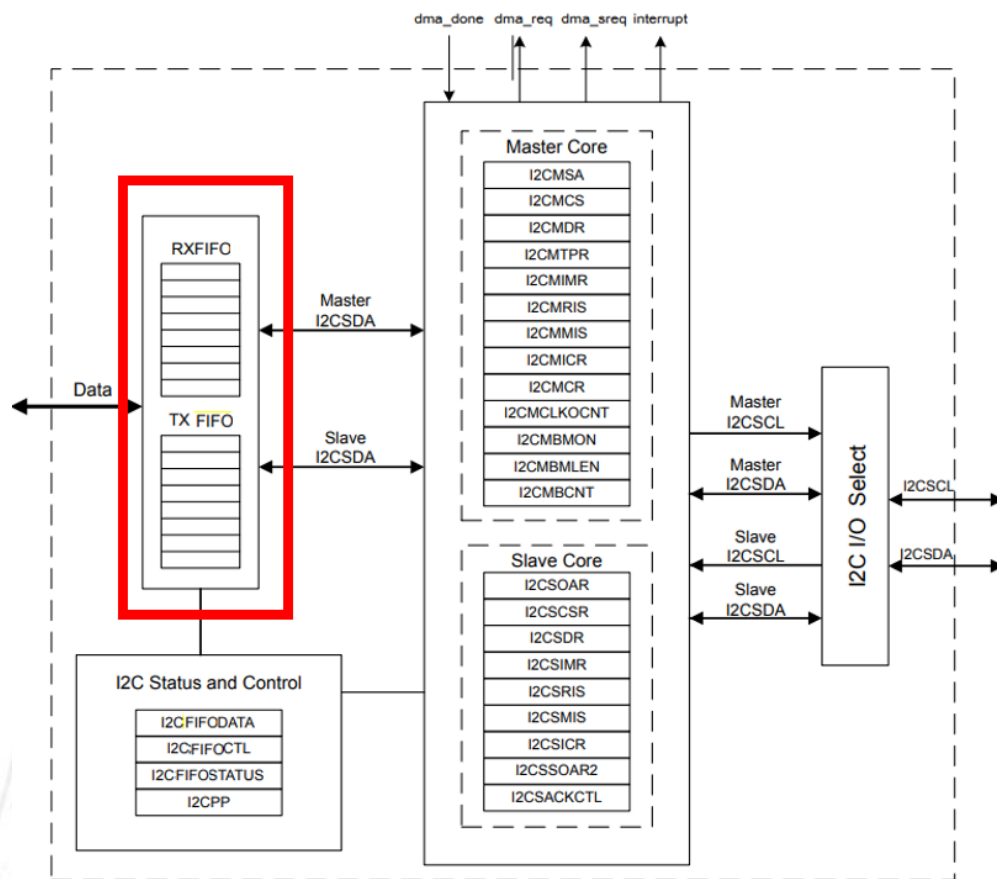
### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

**写**单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能

### TM4C1294的I2C模块的FIFO和突发模式:

- TM4C1294的I2C模块有一个发送FIFO和一个接收FIFO，即可以给主机用，也可以给从机用。通过I2CFIFOCTL寄存器来配置。
- 写I2CFIFODATA寄存器填入数据
- 读取I2CFIFODATA可以读出数据



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能

寄存器I2CFIFODATA

#### I2C FIFO Data (I2CFIFODATA)

I2C 0 base: 0x4002.0000

I2C 1 base: 0x4002.1000

I2C 2 base: 0x4002.2000

I2C 3 base: 0x4002.3000

I2C 4 base: 0x400C.0000

I2C 5 base: 0x400C.1000

I2C 6 base: 0x400C.2000

I2C 7 base: 0x400C.3000

I2C 8 base: 0x400B.8000

I2C 9 base: 0x400B.9000

Offset 0xF00

Type RO, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DATA	RO	0	I <sup>2</sup> C RX FIFO Read Data Byte This field contains the current byte being read in the RX FIFO stack.



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能

### 寄存器I2CFIFOSTATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved													RXABVTRIG	RXFF	RXFE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													TXBLWTRIG	TXFF	TXFE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

### 查看FIFO的状态

RXFF: 读FIFO满

RXFE: 读FIFO空

TXFF: 写FIFO满

TXFE: 写FIFO空



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的**FIFO**功能

使用**I2CFIFODataPut**函数，可以将数据填入**I2CFIFOData**寄存器中

```
void
I2CFIFODataPut(uint32_t ui32Base, uint8_t ui8Data)
{
    ASSERT(_I2CBaseValid(ui32Base));
    while(HWREG(ui32Base + I2C_O_FIFOSTATUS) & I2C_FIFOSTATUS_TXFF)
    {
    }
    HWREG(ui32Base + I2C_O_FIFODATA) = ui8Data;
}
```



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的**FIFO**功能

使用**I2CFIFODataGet**函数，可以将数据从**I2CFIFOData**寄存器中读出：

```
uint32_t
I2CFIFODataGet(uint32_t ui32Base)
{
    ASSERT(_I2CBaseValid(ui32Base));
    while(HWREG(ui32Base + I2C_O_FIFOSTATUS) & I2C_FIFOSTATUS_RXFE)
    {
    }
    // Read a byte.
    //
    return(HWREG(ui32Base + I2C_O_FIFODATA));
}
```





## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

**写单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能**

使用突发模式BURST，可以将FIFO中的数据连续发出，或将从机的数据连续读入到FIFO内。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	WO	WO	WO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	reserved										BURST	QCMD	HS	ACK	STOP	START
											0	0	0	0	0	0

突发模式的传输数据的数量由I2CMBLEN寄存器决定

可以通过I2CMasterBurstLengthSet函数，设置I2CMBLEN寄存器

```
void I2CMasterBurstLengthSet(uint32_t ui32Base, uint8_t ui8Length)
{ HWREG(ui32Base + I2C_O_MBLEN) = ui8Length; }
```



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I<sup>2</sup>C模块的FIFO功能

SINGLE-BYTE WRITE								
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		STOP
SLAVE			ACK		ACK		ACK	

```
I2CTxFIFOFlush(I2C0_BASE);  
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400);};  
while(I2CMasterBusy(I2C0_BASE)){SysCtlDelay(400);};  
I2CFIFODataPut(I2C0_BASE, reg);  
I2CFIFODataPut(I2C0_BASE, value);  
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);  
I2CMasterBurstLengthSet(I2C0_BASE, 2);  
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_FIFO_SINGLE_SEND); //0x46  
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));  
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400);};  
I2CState=I2CMasterErr(I2C0_BASE);
```



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

#### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

读多个寄存器（方法一）：使用I2CMCS寄存器控制逐个读取

MULTIPLE-BYTE READ												
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START'	SLAVE ADDRESS + READ		ACK		NACK	STOP
SLAVE			ACK		ACK			ACK		DATA		DATA

1. 发送起始位和从机地址，写
2. 发送DATA0的地址0x32
3. 发送起始位和从机地址，读
4. 读1个数据，并保存
5. 再读1个数据，并保存
6. 再读1个数据，并保存
7. 再读1个数据，并保存
8. 再读1个数据，并保存
9. 再读1个数据，并保存
10. 发送停止位



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

读多个寄存器（方法一）：使用**I2CMCS**寄存器控制逐个读取

```
int i=0;
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
while(I2CMasterBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);
I2CMasterDataPut(I2C0_BASE, firstRegAddress);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
for(i=0;i<6;i++){
    if(i==0){
        I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CRead);
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); //0x0b
    }else if(i==5){
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); //0x05
    }else{
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); //0x09
    }
    do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
    dataBuffer[i]=I2CMasterDataGet(I2C0_BASE);
    I2CState=I2CMasterErr(I2C0_BASE);
}
```

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0



## ➤ TM4C1294的I<sup>2</sup>C模块的应用：

### – ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

读多个寄存器（方法二）：使用FIFO连续读取数据

### I2CMCS

6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

MULTIPLE-BYTE READ												
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START <sup>1</sup>	SLAVE ADDRESS + READ		ACK		NACK	STOP
SLAVE			ACK		ACK			ACK		DATA		DATA

1. 发送起始位和从机地址，写
2. 发送DATA0的地址0x32
3. 发送起始位和从机地址，读
4. 连续读取6个数据
5. 发送停止位
6. 将数据从FIFO中读出



## ➤ TM4C1294的I<sup>2</sup>C模块的应用:

### - ADXL345三轴加速度传感器

- 通过I<sup>2</sup>C总线操作寄存器的方式

读多个寄存器（方法二）：使用FIFO连续读取数据

```
int i=0;
I2CRxFIFOFlush(I2C0_BASE);
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
while(I2CMasterBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CWrite);
I2CMasterDataPut(I2C0_BASE, firstRegAddress);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
I2CState=I2CMasterErr(I2C0_BASE);
I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, I2CRead);
I2CMasterBurstLengthSet(I2C0_BASE, 6);
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE); //0x46
do{SysCtlDelay(400);}while(I2CMasterBusy(I2C0_BASE));
while(I2CMasterBusBusy(I2C0_BASE)){SysCtlDelay(400)};
I2CState=I2CMasterErr(I2C0_BASE);
while((I2CFIFOStatus(I2C0_BASE)&I2C_FIFO_RX_EMPTY)!=I2C_FIFO_RX_EMPTY){
    dataBuffer[i]=I2CFIFODataGet(I2C0_BASE);
    i++;
}
```





## • TM4C1294的I<sup>2</sup>C模块的应用:

### – ADXL345三轴加速度传感器

将DATA0、DATA1两个八位的数据，合成一个16位的数据

```
x = ((short)readBuffer[1] << 8) + readBuffer[0];  
y = ((short)readBuffer[3] << 8) + readBuffer[2];  
z = ((short)readBuffer[5] << 8) + readBuffer[4];
```

再将这个16位的数据转换为加速度，便可以根据重力加速度的分布，获知传感器的倾角状态。





---

## • 小结

- I<sup>2</sup>C总线的基本概念
- I<sup>2</sup>C总线的通讯
- TM4C1294的I2C模块的功能
- TM4C1294的I2C模块的使用
- TM4C1294的I2C模块的应用



作业:

下图是I<sup>2</sup>C总线上的数据，写出具体通信内容，SDA哪部分是主机发的，哪部分是从机发的？

