IIC 通信2

程晨闻

东南大学电气工程学院

➢ 实验任务

– 用PN0和PN1驱动开发板上的两个LED灯，PN0控制D2，PN1控制D1。PJ0和PJ1接两个简单按键，USR_SW1和USR_SW2。

– 基本要求：

  • 每按一次USR_SW1，D2快闪三次（间隔0.33s）。每按一次USR_SW2，D1慢闪三次（间隔2s）。（60分）

– 发挥要求1：

  • D2和D1的闪烁控制不相互影响，即在D1闪烁的时候按USR_SW1，D2快闪的功能不变（20分）

– 发挥要求2：

  • D1和D2的闪烁的次数，分别用变量cD1和cD2表示，可以通过调试界面观察。要求用这两个变量记录D1和D2的闪烁的次数，即使重启，数据依然能够保存。（20分）

➢ 经验总结

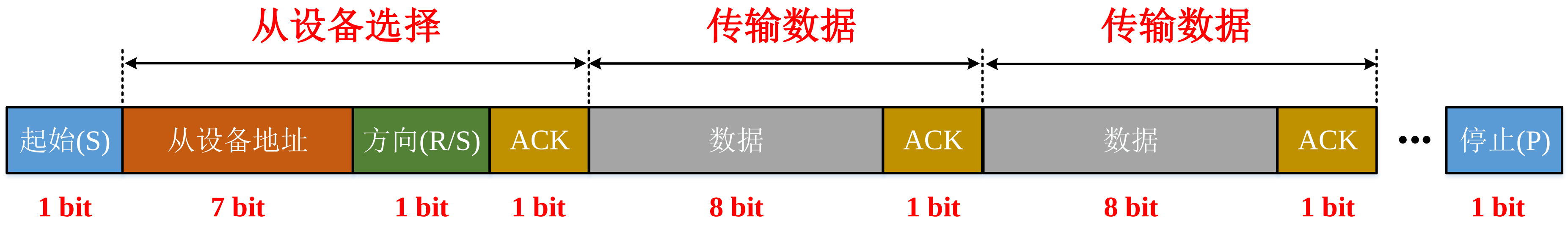
- 独立完成实验，是对自身能力的最好锻炼
- 在例程的基础上修改，不要从头开始
- 重在理解课堂上讲的知识，在理解的基础上加以应用
- 先构思好程序架构，然后实现细节
- 每增加一段代码，编译测试，切勿图快
- 学会**debug**，让错误自己跑出来（**LED**，观察变量）
- 多在开发板上练习，顺能生巧

➢ 内容概要

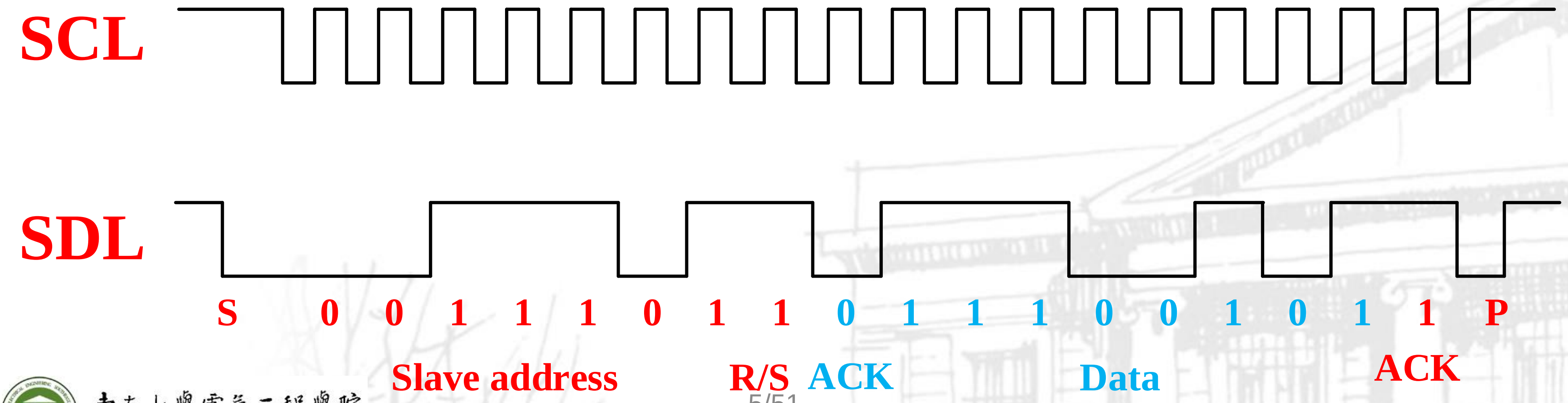
- 通信的基本概念
  - 单工，半双工，全双工
  - 串行，并行
- I2C总线的基本概念
  - 双向、二线制（SDA 和SCL）、同步、串行
  - 开漏（线与）
- I2C总线的通讯
  - 起始位和结束位
  - 从机地址（7 bits）+ R/S（1 bit）+ ACK
  - Data （8 bits）+ ACK

作业

➢ 从从机0x1D读一个数据，如果这个数据是0xE5， 画出SCL和SDA的波形？标出那部分波形是主机发送的，哪部分是从机发送的。



0x1D：0001 1101　　0xE5：1110 0101　　R/S = 1



➢ 内容概要

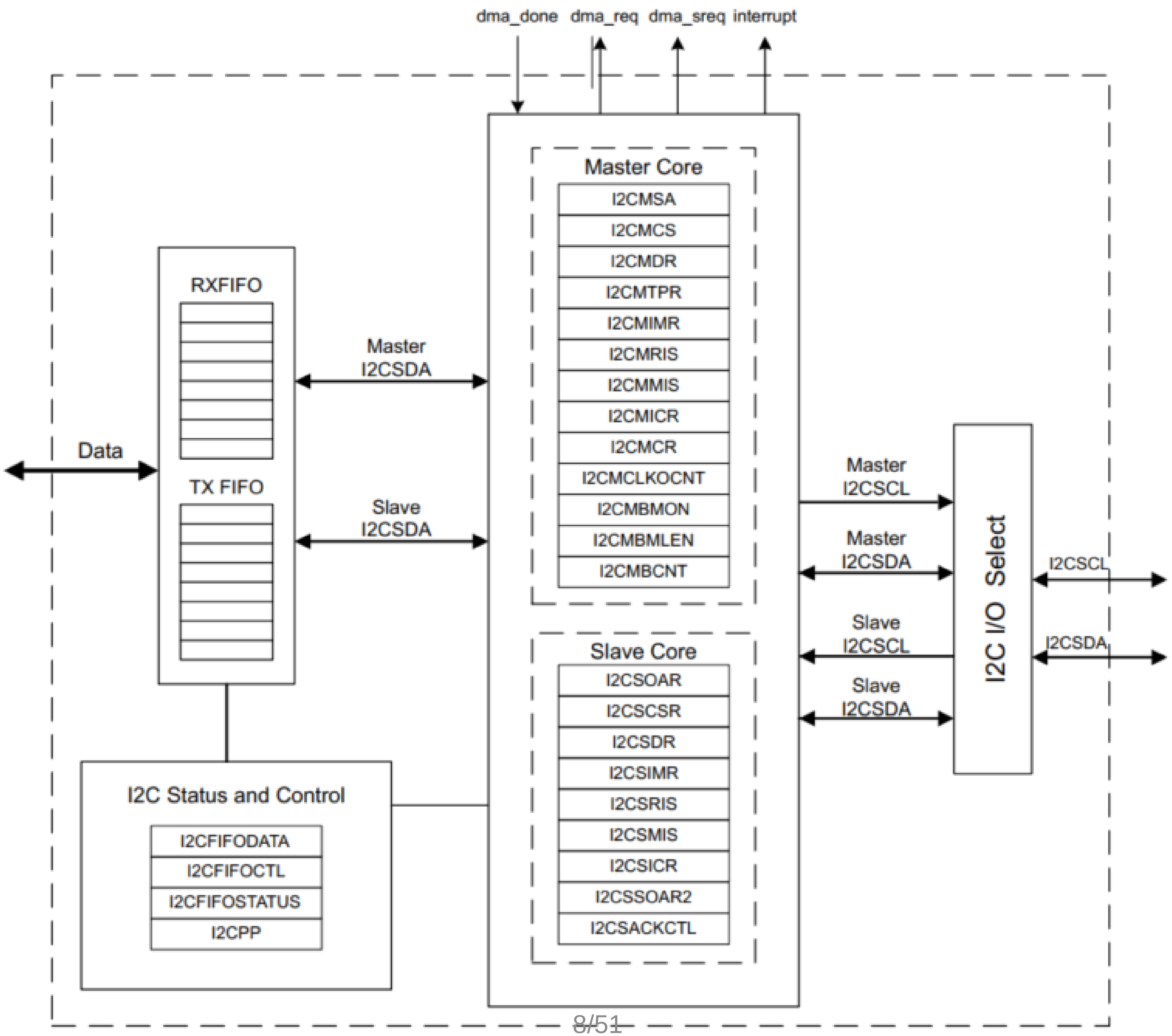
– TM4C1294的I2C模块的功能

– TM4C1294的I2C模块的使用

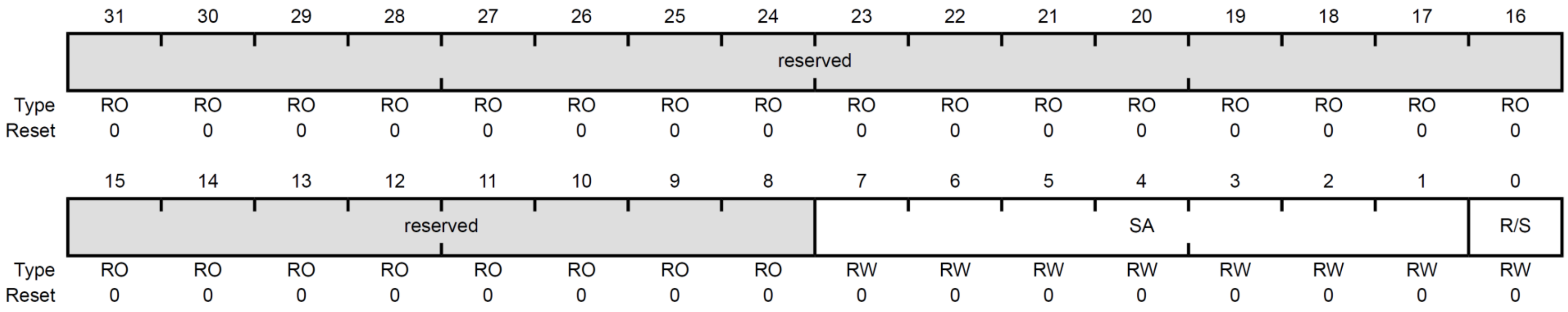
➢ TM4C的I2C模块

- 支持**主从**/**收发**模式
- 发送，接收都有**FIFO**
  - FIFO(First Input First Output)，先进先出
  - 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
  - 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，从而提高了系统性能
- 支持四种传输速度
  - Standard (**100K** bps)
  - Fast-mode (**400K** bps)
  - Fast-mode plus (**1M** bps)
  - High-speed mode (**3.33M** bps)
- 支持发送接收**中断**
- 支持DMA

TM4C的I2C模块


dma_done
dma_req
dma_sreq
interrupt
Data
RXFIFO
TX FIFO
Master I2CSDA
Slave I2CSDA
Master Core
I2CMSA
I2CMCS
I2CMDR
I2CMTPR
I2CMIMR
I2CMRIS
I2CMMIS
I2CMICR
I2CMCR
I2CMCLKOCNT
I2CMBMON
I2CMBMLEN
I2CMBCNT
Slave Core
I2CSOAR
I2CSCSR
I2CSDR
I2CSIMR
I2CSRIS
I2CSMIS
I2CSICR
I2CSSOAR2
I2CSACKCTL
I2C Status and Control
I2CFIFODATA
I2CFIFOCTL
I2CFIFOSTATUS
I2CPP
Master I2CSCL
Master I2CSDA
Slave I2CSCL
Slave I2CSDA
I2C I/O Select
I2CSCL
I2CSDA

➢ I2C Master Slave Address (I2CMSA)



	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								SA							R/S
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	0	0	0	0	0	0	0	0

地址寄存器，最低位**R/S**表示接收/发送

R/S = 0, 发送

R/S = 1, 接收

➢ I2C Master Control/Status (I2CMCS) 读

	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	DATACK	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

BUSBSY： 0，总线空闲
1， 总线忙

DATACK： 0，发送数据被应答
1，发送数据未被应答

ADPACK： 0，地址被应答
1，地址未被应答

BUSY： 0，控制器空闲
1，控制器忙

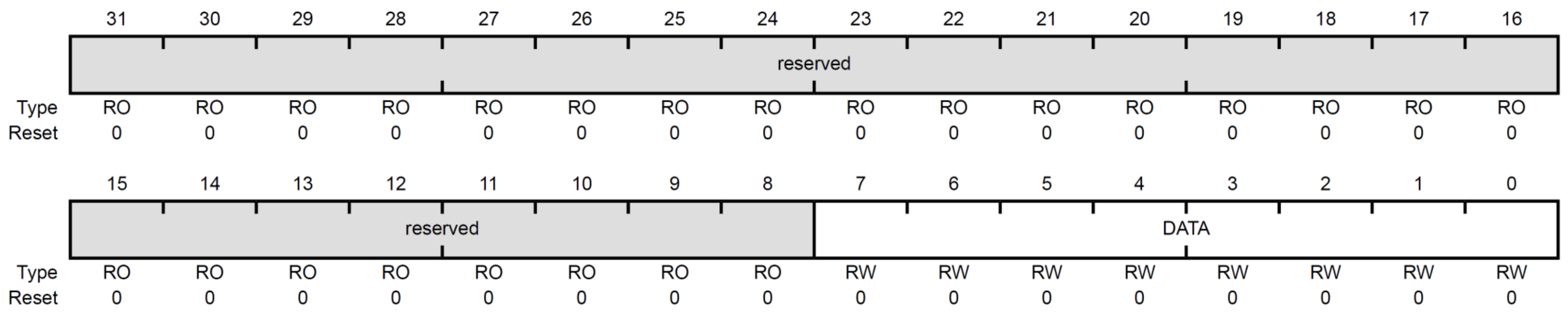
➢ I2C Master Control/Status (I2CMCS) 写

	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

HS:	0，标准，快速，超快速（由I2CMTPR决定）
	1，高速
ACK:	0，主机不自动应答接收数据
	1，主机自动应答接收数据
STOP:	0，控制器不产生停止位
	1，控制器产生停止位
START:	0，控制器不产生开始位
	1，控制器产生开始位或重复开始位
RUN:	0，不发送或接收数据
	1，使能发送或接收数据

➢ I2C Master Data (I2CMDR)



	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

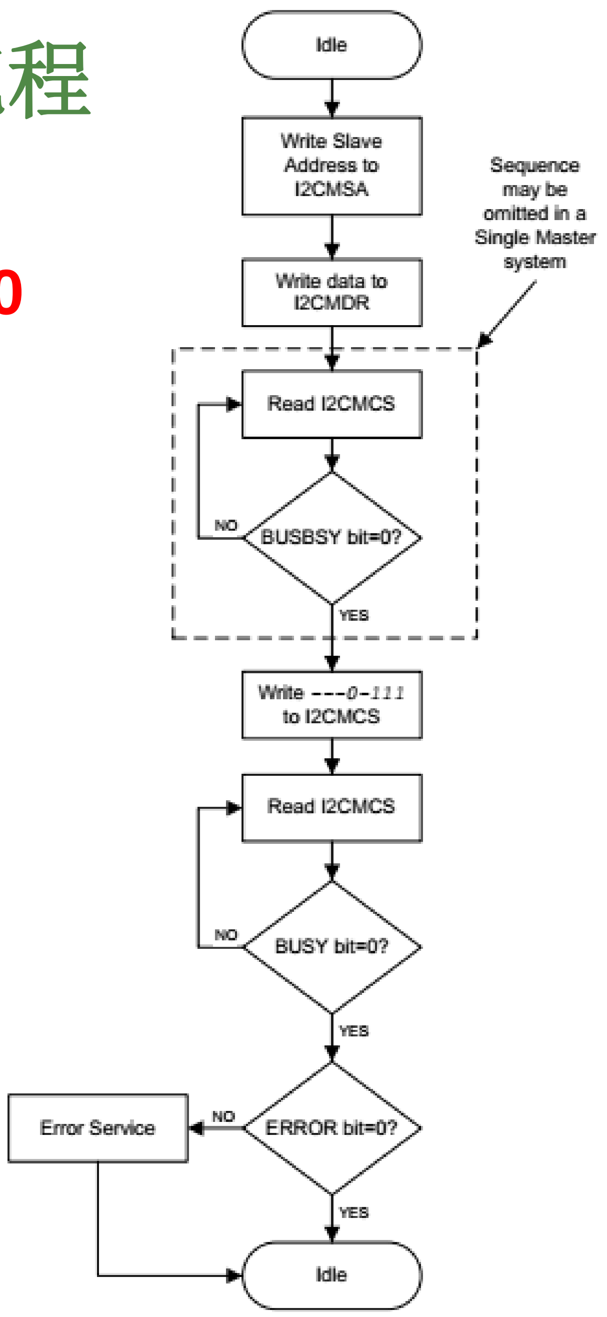
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DATA： 包含接收或要发送的数据（主机模式）

➢ 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**， **R/S = 0**
- 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

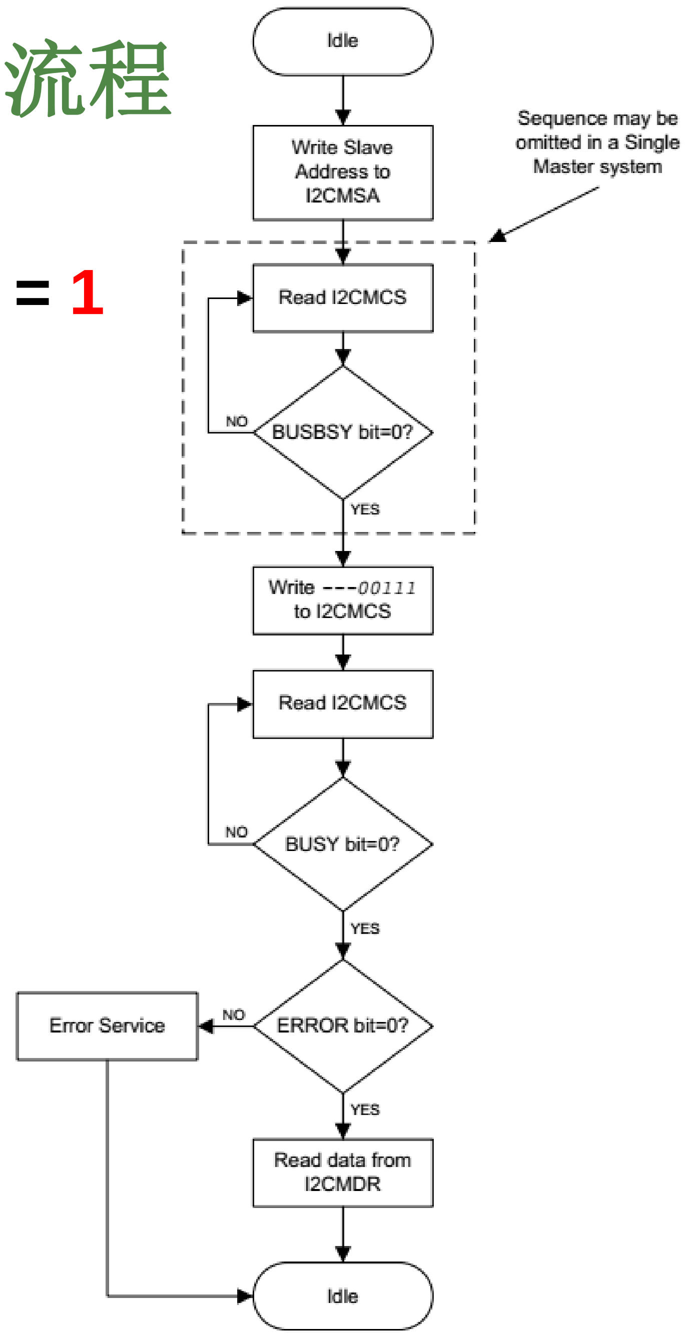


6	5	4	3	2	1	0
BURST	QCMD	HS	ACK	STOP	START	RUN
WO	WO	WO	WO	WO	WO	WO
0	0	0	0	0	0	0

发送和接收数据的具体操作流程

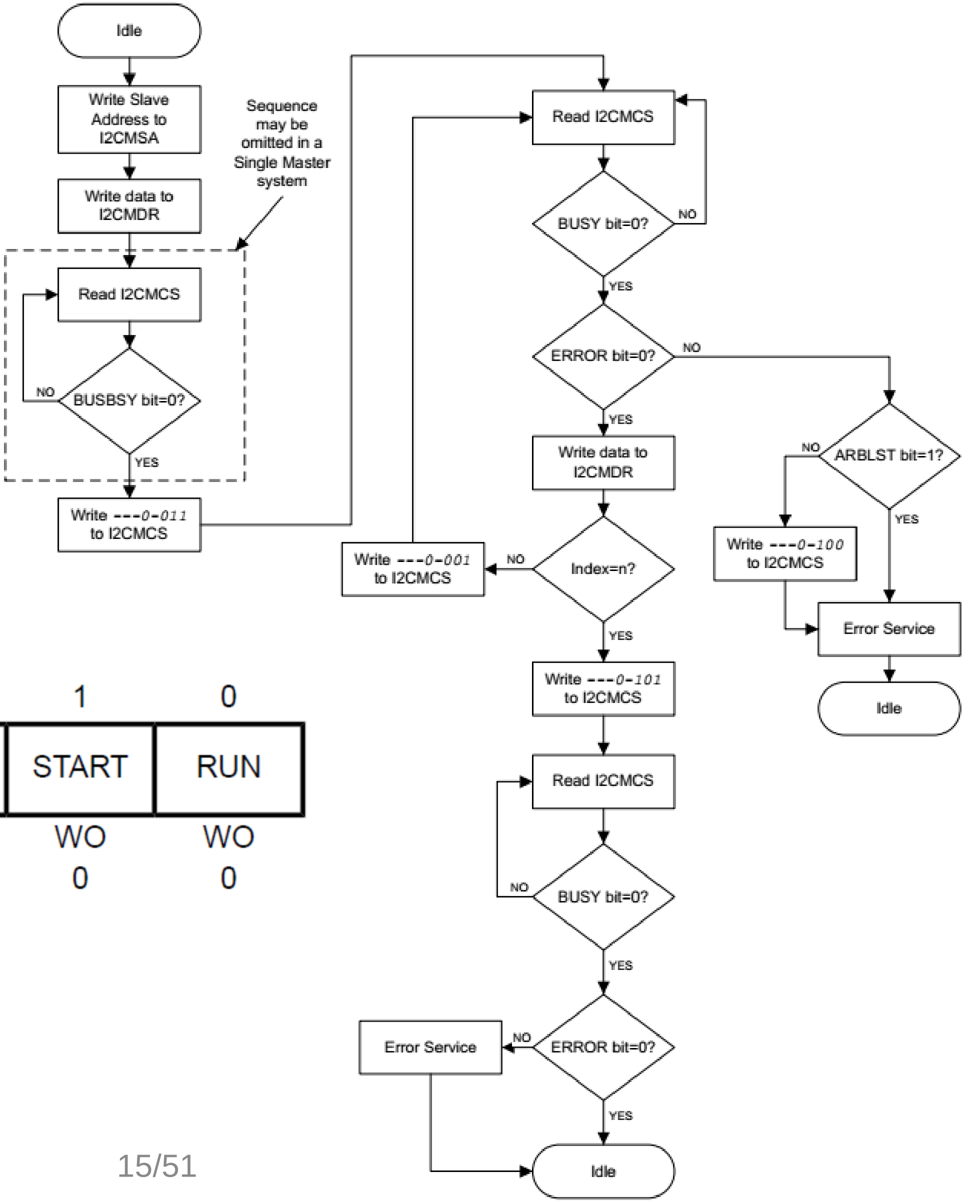
– 2. 主设备读单个数据：
  - 将**从机地址**写入**I2CMSA**， **R/S = 1**
  - 检查总线是否忙碌
  - 将---**00111**写入**I2CMCS**
  - 等到控制器忙碌结束
  - 检查是否有错误
  - **从I2CMDR读出数据**

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



➢ 发送和接收数据的具体操作流程

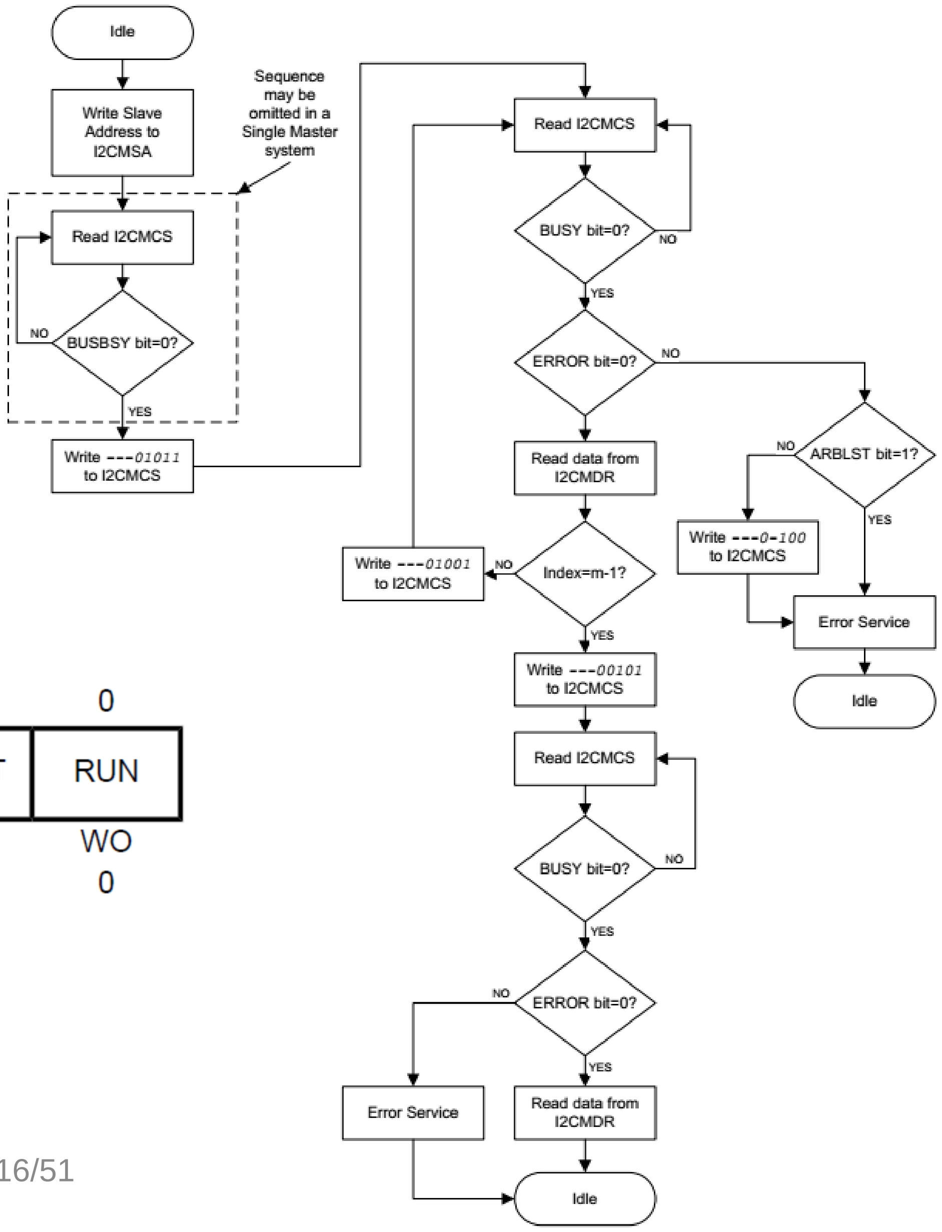
– 3. 写多个数据：



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

➢ 发送和接收数据的具体操作流程

– 4. 读多个数据：



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

➢ 作业

– 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I2C模块的使用(主机发送)

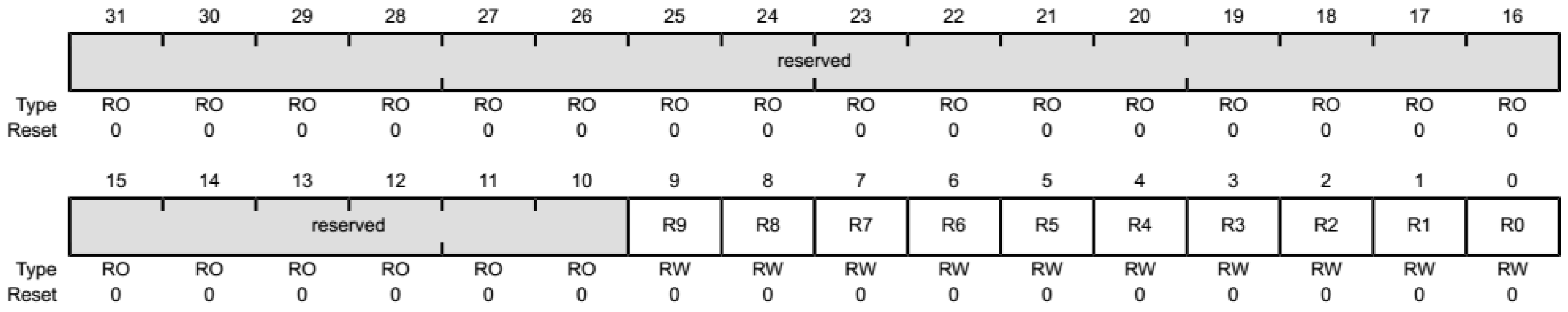
- 1. 在系统控制模块中，设置RCGCI2C寄存器，使能所需要用到的I2C模块
- 2. 在系统控制模块中，使能总线所在的GPIO模块的时钟
- 3. 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的复用功能
- 4. 设置I2CSDA引脚为漏极开路
- 5. 配置GPIOPCTL的PMCn位，将GPIO模块相应引脚的信号连接至I2C模块
- 6. 向I2CMCR寄存器中写0x00000010 初始化I2C模块
- 7. 配置I2CMTPR寄存器，设置I2C模块的时钟
- 8. 写I2CMSA寄存器，设置目标从机的地址，设置R/S
- 9. 将要发送的数据写入I2CMDR寄存器
- 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I2C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I2C模块

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

Base 0x400F.E000
Offset 0x620
Type RW, reset 0x0000.0000



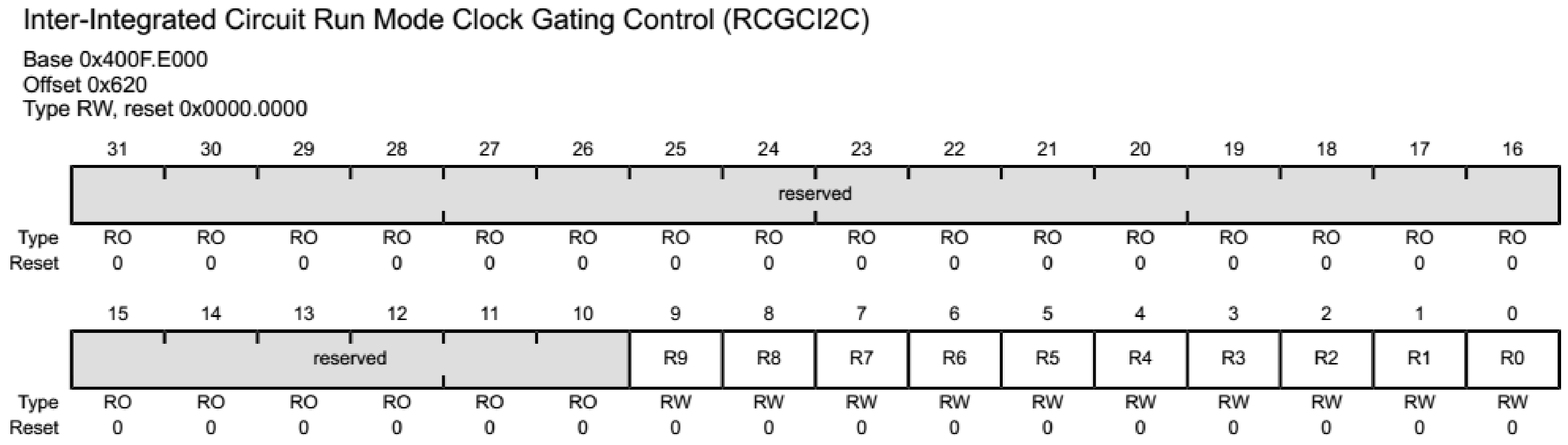
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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

I2C模块时钟控制寄存器，控制10个I2C模块的时钟，R0至R9分别控制I2C0模块、I2C1模块至I2C9模块。

➢ TM4C1294的I2C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I2C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

方法1：

直接使用

HWREGBITW(0x400FE000+0x620, 0) = 1;

来操作该寄存器对应的位

➢ TM4C1294的I2C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I2C模块（如I2C0）

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I2C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^{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.

  - 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I2C模块的使用

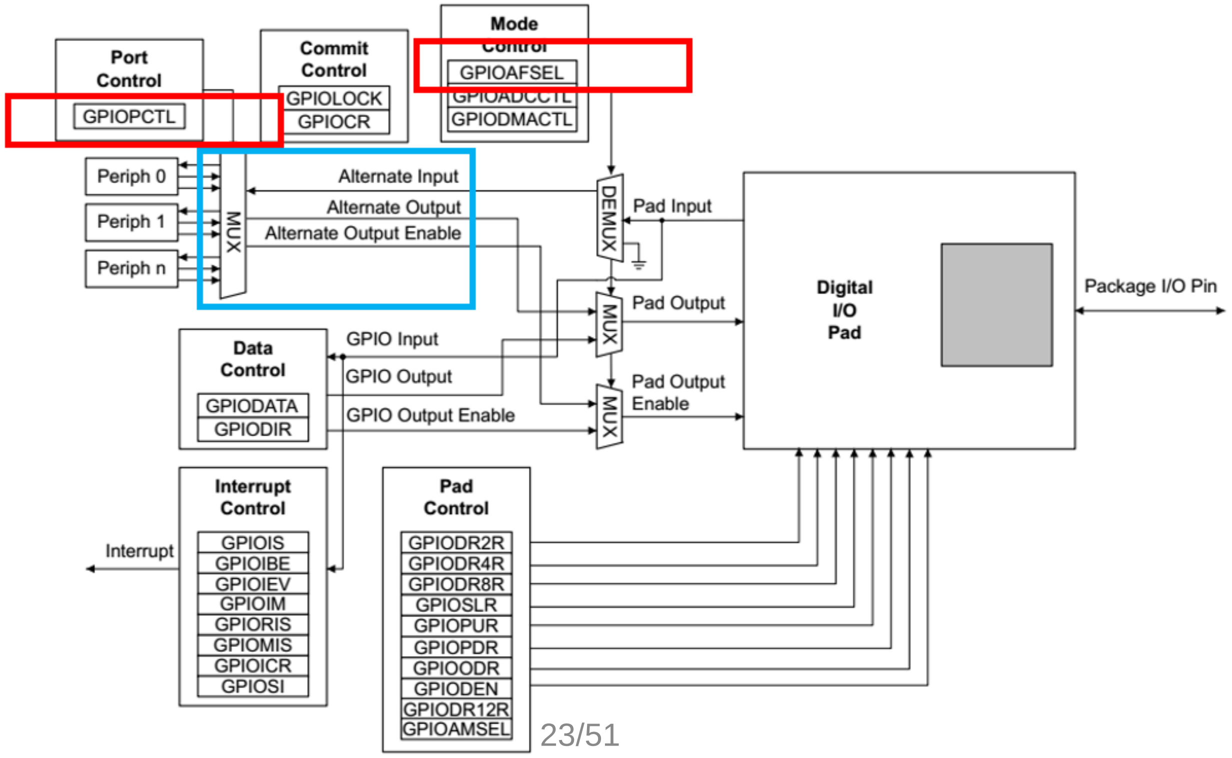
– 3. 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的复用功能

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**GPIOPCTL**决定



➢ TM4C1294的I2C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**GPIOPCTL**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
	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								AFSEL							
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	-	-	-	-	-	-	-	-

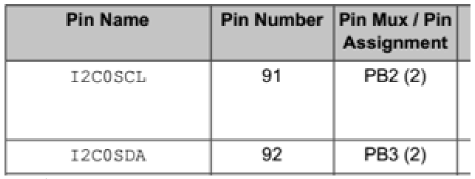
Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	AFSEL	RW	-	GPIO Alternate Function Select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

```
HWREG(0x40059000 + 0x420) = ( (HWREG(0x40059000 + 0x420) |0x0C));
```

東南大學電氣工程學院
SCHOOL OF ELECTRICAL ENGINEERING, SEU

➢ TM4C1294的I2C模块的使用

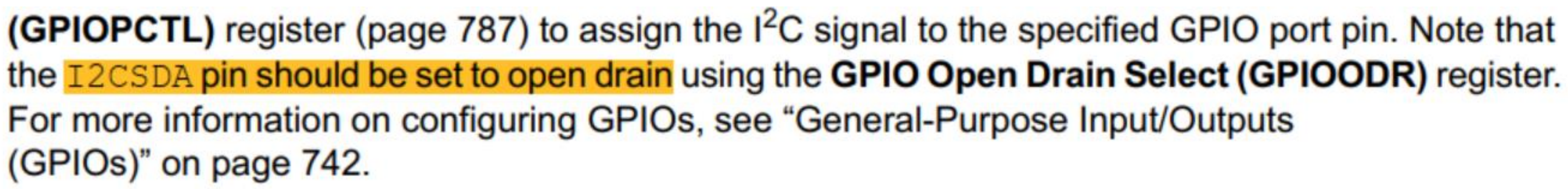
- 3. 设置GPIO模块的GPIOAFSEL寄存器，配置GPIO的**复用功能**
- 4. 设置I2CSDA引脚为**漏极开路**



Pin Name	Pin Number	Pin Mux / Pin Assignment
I2C0SCL	91	PB2 (2)
I2C0SDA	92	PB3 (2)

调用GPIOPinTypeI2C和GPIOPinTypeI2CSCL函数:

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



(GPIOPCTL) register (page 787) to assign the I^{2}C signal to the specified GPIO port pin. Note that the I2CSDA pin should be set to open drain using the **GPIO Open Drain Select (GPIOODR)** register. For more information on configuring GPIOs, see "General-Purpose Input/Outputs (GPIOs)" on page 742.

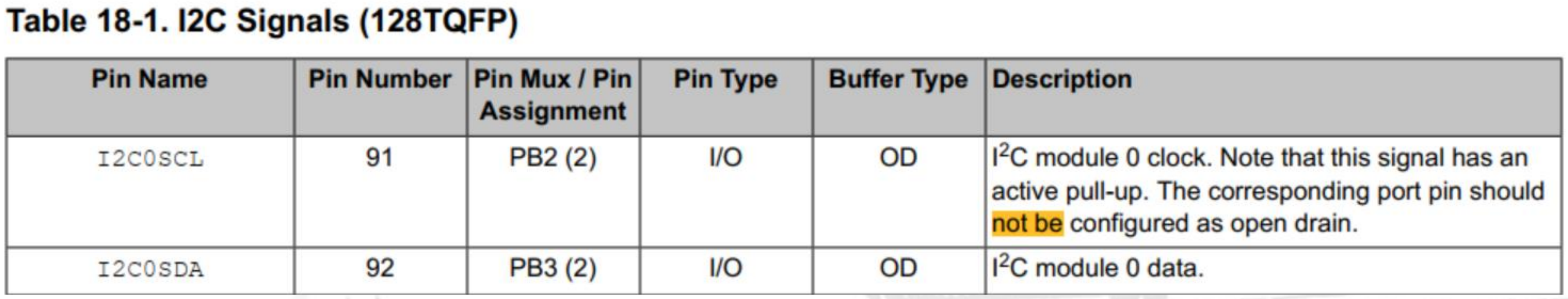
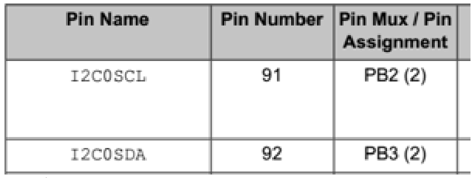


Table 18-1. I2C Signals (128TQFP)

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.

➢ TM4C1294的I2C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I2CSDA引脚为漏极开路



Pin Name	Pin Number	Pin Mux / Pin Assignment
I2C0SCL	91	PB2 (2)
I2C0SDA	92	PB3 (2)

调用GPIOPinTypeI2C和GPIOPinTypeI2CSCL函数:

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

```
#define GPIO_DIR_MODE_HW        0x00000002  // Pin is a peripheral function
```

```
void
GPIOPinTypeI2C(uint32_t ui32Port, uint8_t ui8Pins)
{
    …
    // Make the pin(s) be peripheral controlled.
    //
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);

    //
    // Set the pad(s) for open-drain operation with a weak pull-up.
    //
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_OD);
}
```

➢ TM4C1294的I2C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I2CSDA引脚为**漏极开路**

```
#define GPIO_DIR_MODE_HW        0x00000002  // Pin is a peripheral function
```

```
void
GPIODirModeSet(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I2C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**I2CSDA**引脚为**漏极开路**

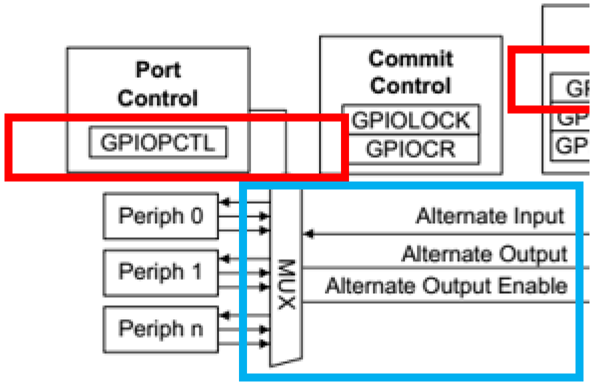
  调用GPIOPinTypeI2C和GPIOPinTypeI2CSCL函数：

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

```
void
GPIOPinTypeI2CSCL(uint32_t ui32Port, uint8_t ui8Pins)
{
    …
    // Make the pin(s) be peripheral controlled.
    //
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);

    //
    // Set the pad(s) for push-pull operation.
    //
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);
}
```

➢ TM4C1294的I2C模块的使用

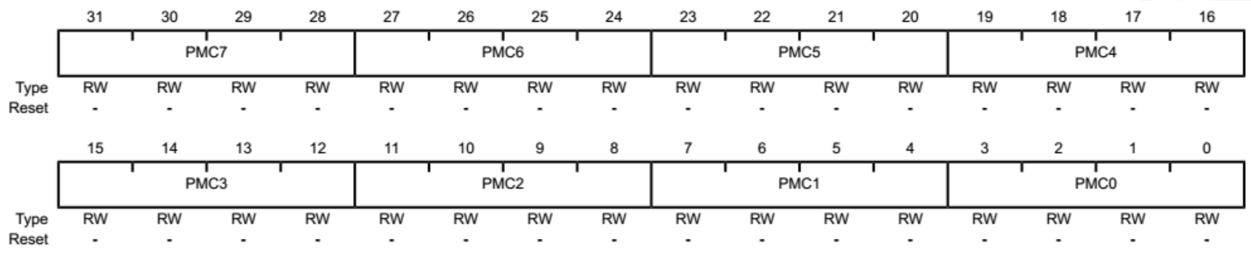
– 5. 配置GPIOPCTL的PMCn位，将GPIO模块相应引脚的信号连接至I2C模块



GPIOPCTL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，分别代表不同的复用功能。

GPIO Port Control (GPIOPCTL)

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



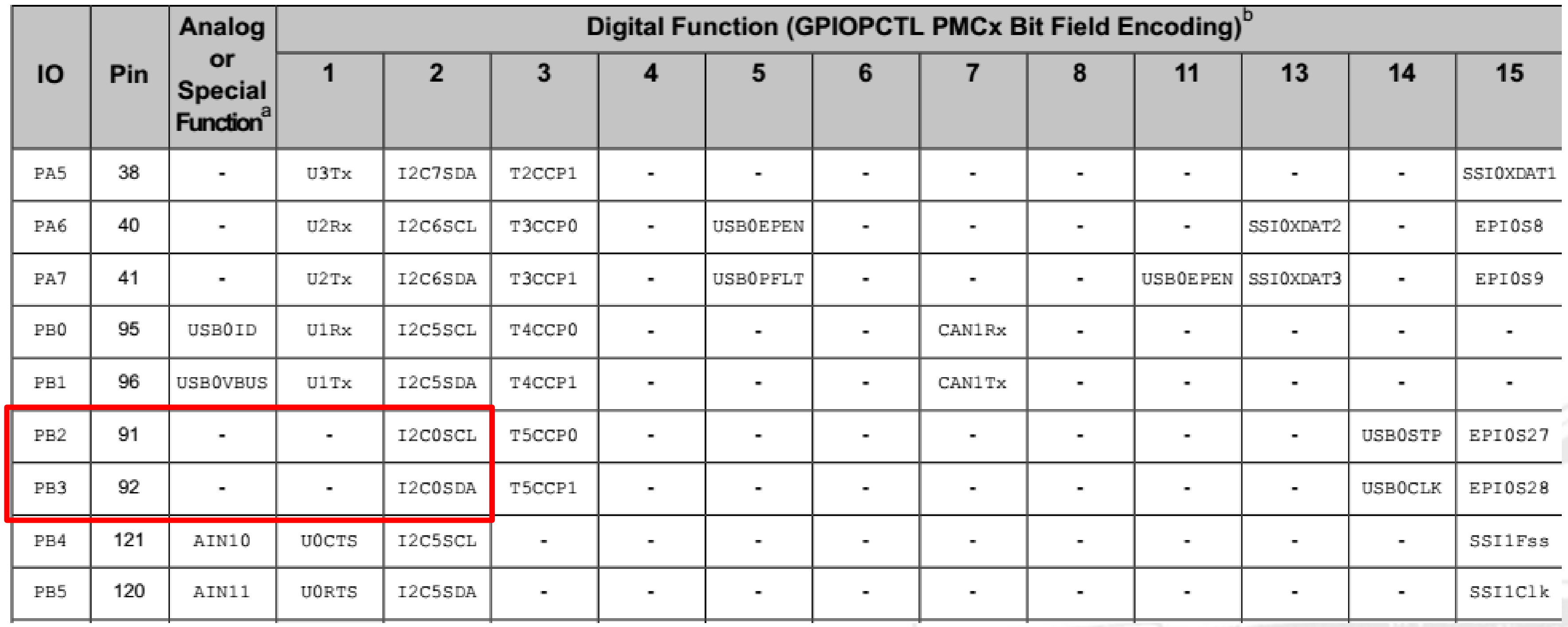
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PMC7				PMC6				PMC5				PMC4			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	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I2C模块的使用

– 5. 配置GPIOPCTL的PMCn位，将GPIO模块相应引脚的信号连接至I2C模块

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



IO	Pin	Analog or Special Function[a]	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[b]											
			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PMC2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PMC3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I2C模块的使用

- 5. 配置GPIOPCTL的PMCn位，将GPIO模块相应引脚的信号连接至I2C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I2C模块的使用

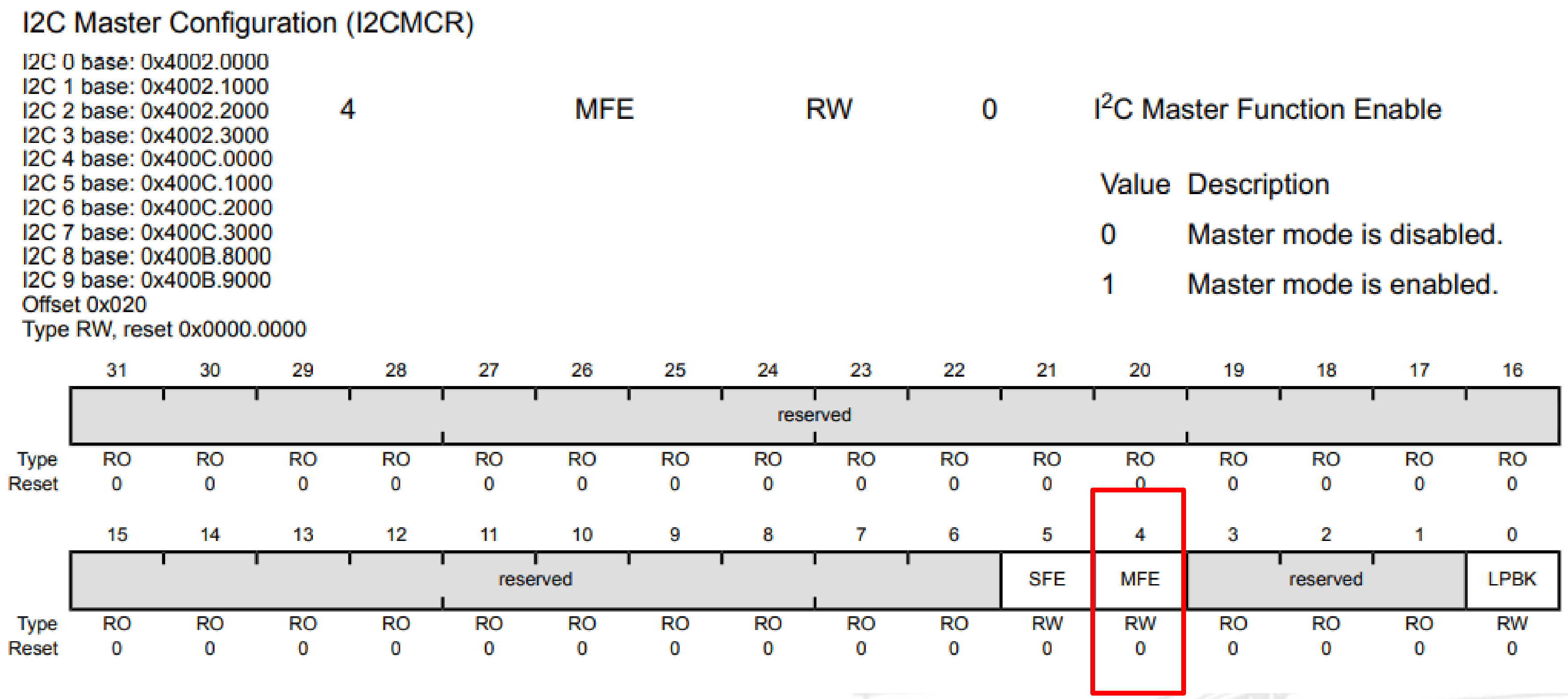
– 5. 配置GPIOPCTL的PMCn位，将GPIO模块相应引脚的信号连接至I2C模块

```
#define GPIO_PB2_I2C0SCL        0x00010802
#define GPIO_PB3_I2C0SDA        0x00010C02
```

```
void
GPIOPinConfigure(uint32_t ui32PinConfig)
{
    uint32_t ui32Base, ui32Shift;
    …
    // Extract the base address index from the input value.
    ui32Base = (ui32PinConfig >> 16) & 0xff;
    // Get the base address of the GPIO module, selecting either the APB or the
    // AHB aperture as appropriate.
    if(HWREG(SYSCTL_GPIOHBCTL) & (1 << ui32Base))
    {
        ui32Base = g_pui32GPIOBaseAddrs[(ui32Base << 1) + 1];
    }
    else
    {
        ui32Base = g_pui32GPIOBaseAddrs[ui32Base << 1];
    }
    // Extract the shift from the input value.
    ui32Shift = (ui32PinConfig >> 8) & 0xff;
    // Write the requested pin muxing value for this GPIO pin.
    HWREG(ui32Base + GPIO_O_PCTL) = ((HWREG(ui32Base + GPIO_O_PCTL) &
                                      ~(0xf << ui32Shift)) |
                                     ((ui32PinConfig & 0xf) << ui32Shift));
}
```

➢ TM4C1294的I2C模块的使用

– 6. 向I2CMCR寄存器中写0x00000010 初始化I2C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

4	MFE	RW	0	I^{2}C Master Function Enable Value Description 0 Master mode is disabled. 1 Master mode is enabled.

	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										SFE	MFE	reserved			LPBK
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RO	RO	RO	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

直接操作寄存器，使能主机功能：
HWREG(I2C0_BASE+ I2C_O_MCR) = 0x00000010;

➢ TM4C1294的I2C模块的使用

– 6. 向I2CMCR寄存器中写0x00000010 **初始化**I2C模块

```
#define I2C_MCR_MFE         0x00000010  // I2C Master Function Enable
```

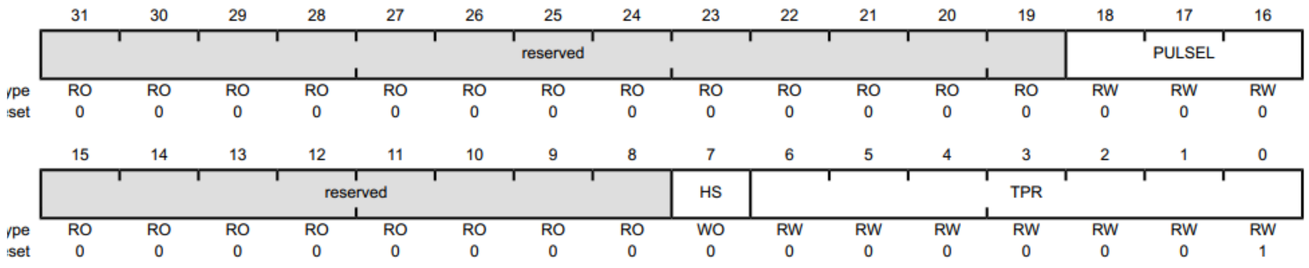
```
void
I2CMasterEnable(uint32_t ui32Base)
{
    //
    // Check the arguments.
    //
    ASSERT(_I2CBaseValid(ui32Base));

    //
    // Enable the master block.
    //
    HWREG(ui32Base + I2C_O_MCR) |= I2C_MCR_MFE;
}
```

➢ TM4C1294的I2C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		
ype	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
set	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						
ype	RO	RO	RO	RO	RO	RO	RO	RO	WO	RW	RW	RW	RW	RW	RW	RW
set	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

6:0	TPR	RW	0x1	Timer Period

This field is used in the equation to configure SCL_PERIOD:

$$SCL\_PERIOD = 2 \times (1 + TPR) \times (SCL\_LP + SCL\_HP) \times CLK\_PRD$$

where:

SCL_PRD is the SCL line period (I^2C clock).

TPR is the Timer Period register value (range of 1 to 127).

SCL_LP is the SCL Low period (fixed at 6).

SCL_HP is the SCL High period (fixed at 4).

CLK_PRD is the system clock period in ns.

➢ TM4C1294的I2C模块的使用

– 7. 配置**I2CMTPR**寄存器，设置I2C模块的**时钟**

	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

TPR区域决定了I2C的时钟

TPR = (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I2C时钟

如果系统时钟为120MHz，使用100kHz的I2C时钟，那么TPR的值为：

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

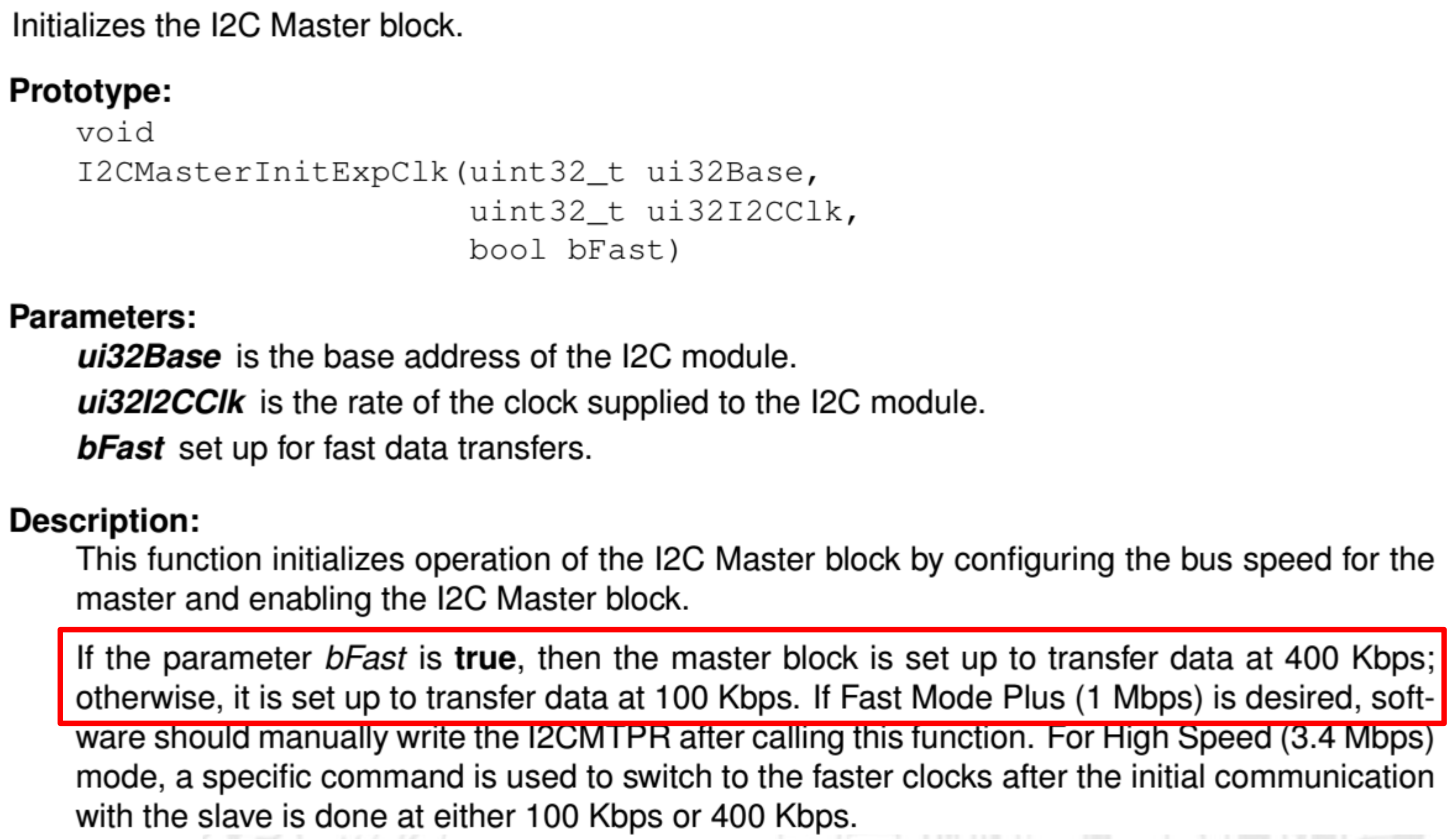
计算出TPR的值后，操作寄存器设置I2CMTPR寄存器：

HWREG(I2C0_BASE + I2C_O_MTPR) = 59;

➢ TM4C1294的I2C模块的使用

- 6. 向**I2CMCR**寄存器中写**0x00000010** **初始化**I2C模块
- 7. 配置**I2CMTPR**寄存器，设置I2C模块的**时钟**

TivaWare库提供了函数**I2CmasterInitExpClk**可以一次性初始化I2C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I2C模块的使用

- 6. 向I2CMCR寄存器中写0x00000010 **初始化**I2C模块
- 7. 配置I2CMTPR寄存器，设置I2C模块的**时钟**

TivaWare库提供了函数I2CmasterInitExpClk可以一次性初始化I2C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。

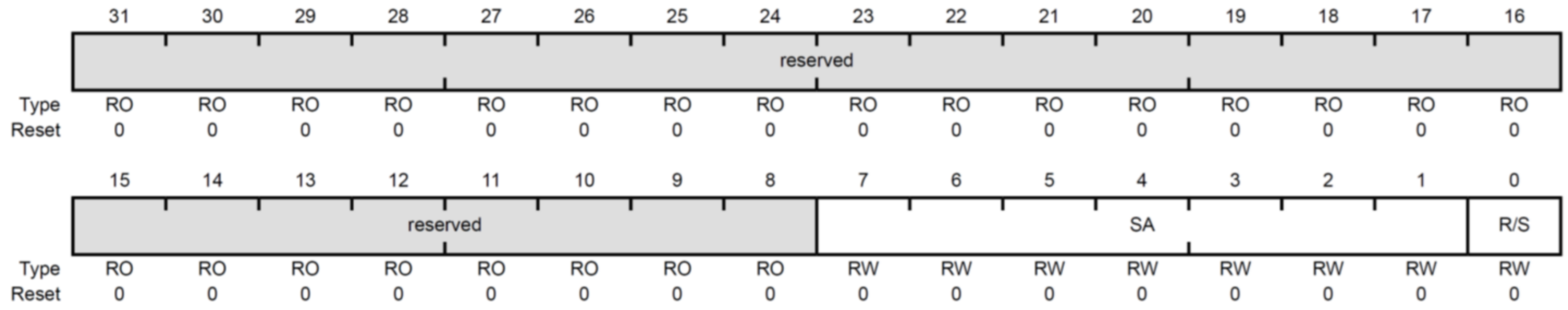
I2C模块的使用

```
void
I2CMasterInitExpClk(uint32_t ui32Base, uint32_t ui32I2CClk, bool bFast)
{
    uint32_t ui32SCLFreq;
    uint32_t ui32TPR;
    …
    // Must enable the device before doing anything else.
    I2CMasterEnable(ui32Base);
    // Get the desired SCL speed.
    if(bFast == true) {ui32SCLFreq = 400000;}
    else {ui32SCLFreq = 100000;}
    // Compute the clock divider that achieves the fastest speed less than or
    // equal to the desired speed.  The numerator is biased to favor a larger
    // clock divider so that the resulting clock is always less than or equal
    // to the desired clock, never greater.
    ui32TPR = ((ui32I2CClk + (2 * 10 * ui32SCLFreq) - 1) /
               (2 * 10 * ui32SCLFreq)) - 1;
    HWREG(ui32Base + I2C_O_MTPR) = ui32TPR;
    // Check to see if this I2C peripheral is High-Speed enabled.  If yes, also
    // choose the fastest speed that is less than or equal to 3.4 Mbps.
    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;
    }
}
```

➢ TM4C1294的I2C模块的使用

– 8. 写I2CMSA寄存器，设置目标从机的地址，设置R/S

I2C Master Slave Address (I2CMSA)



	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								SA							R/S
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	0	0	0	0	0	0	0	0

方法1:
直接操作寄存器设置I2CMSA寄存器：HWREG(I2C0_BASE + I2C_O_MSA) = ????;

地址寄存器，最低位**R/S**表示接收/发送

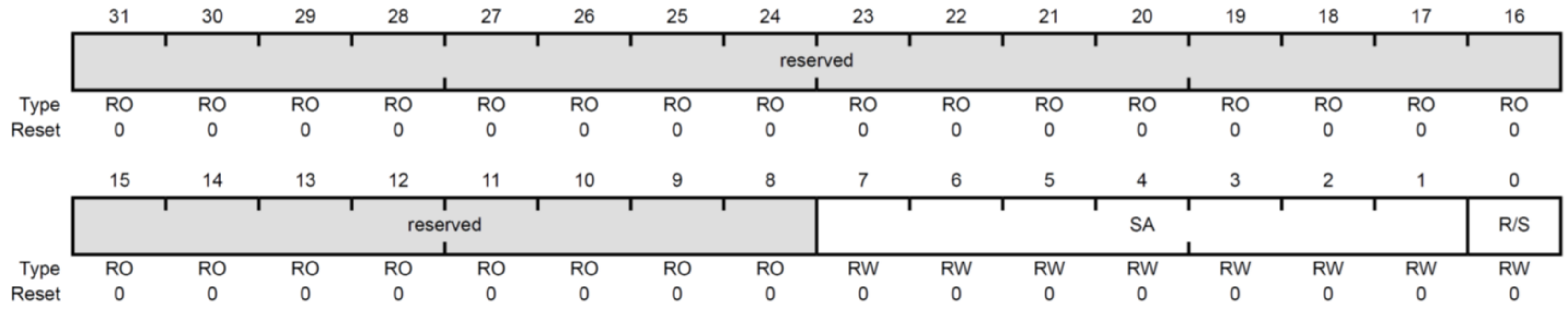
R/S = 0, 发送

R/S = 1, 接收

➢ TM4C1294的I2C模块的使用

– 8. 写I2CMSA寄存器，设置目标从机的地址，设置R/S

I2C Master Slave Address (I2CMSA)



	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								SA							R/S
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	0	0	0	0	0	0	0	0

方法2：使用TivaWare库函数I2CMasterSlaveAddrSet设置I2CMSA寄存器。

```c
void
I2CMasterSlaveAddrSet(uint32_t ui32Base, uint8_t ui8SlaveAddr, bool bReceive)
{
    // Check the arguments.
    ASSERT(_I2CBaseValid(ui32Base));
    ASSERT(!(ui8SlaveAddr & 0x80));
    // Set the address of the slave with which the master will communicate.
    //
    HWREG(ui32Base + I2C_O_MSA) = (ui8SlaveAddr << 1) | bReceive;
}
```

➢ TM4C1294的I2C模块的使用

– 9. 将要发送的**数据**写入I2CMDR寄存器

I2C Master Data (I2CMDR)

	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	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

调用TivaWare库的**I2CMasterDataPut**函数

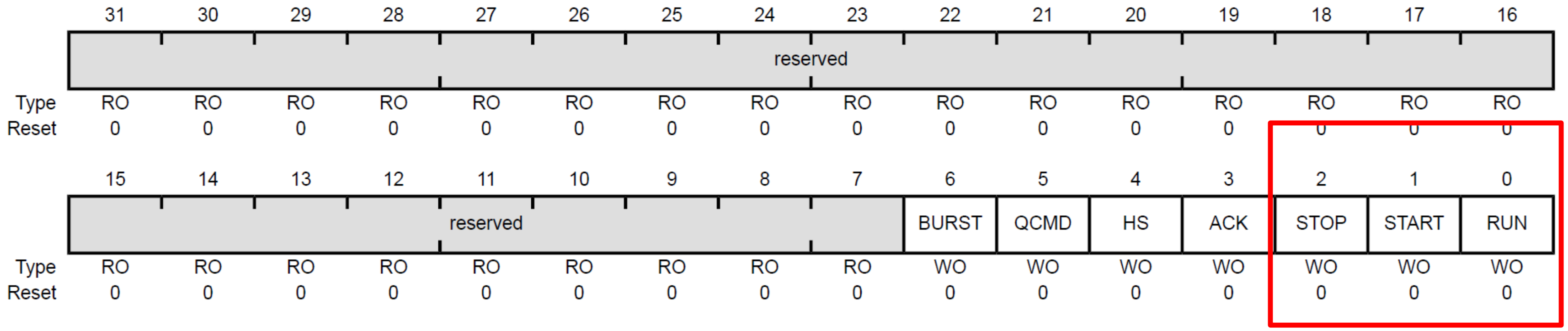
```
void
I2CMasterDataPut(uint32_t ui32Base, uint8_t ui8Data)
{
    // Check the arguments.
    //
    ASSERT(_I2CBaseValid(ui32Base));
    // Write the byte.
    //
    HWREG(ui32Base + I2C_O_MDR) = ui8Data;
}
```

➢ TM4C1294的I2C模块的使用

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

I2C 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

STOP：	0，	控制器不产生停止位
	1，	控制器产生停止位
START：	0，	控制器不产生开始位
	1，	控制器产生开始位或重复开始位
RUN：	0，	不发送或接收数据
	1，	使能发送或接收数据

➢ TM4C1294的I2C模块的使用

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

I2C 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

```
Void I2CMasterControl(uint32_t ui32Base, uint32_t ui32Cmd)
{
    // Check the arguments.
    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));
    // Send the command.
    HWREG(ui32Base + I2C_O_MCS) = ui32Cmd;
}
```

I2C模块的使用

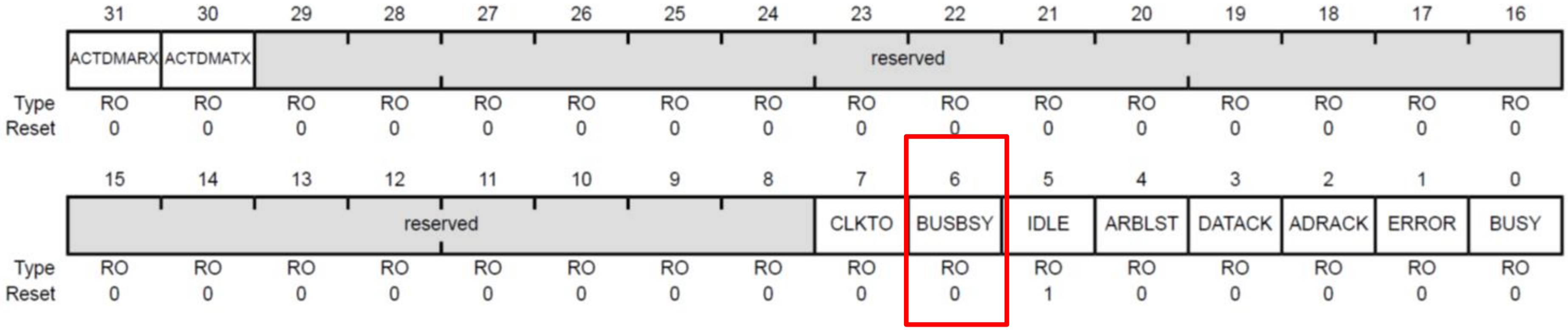
```
#define I2C_MASTER_CMD_SINGLE_SEND                  0x00000007
#define I2C_MASTER_CMD_SINGLE_RECEIVE               0x00000007
#define I2C_MASTER_CMD_BURST_SEND_START             0x00000003
#define I2C_MASTER_CMD_BURST_SEND_CONT              0x00000001
#define I2C_MASTER_CMD_BURST_SEND_FINISH            0x00000005
#define I2C_MASTER_CMD_BURST_SEND_STOP              0x00000004
#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
#define I2C_MASTER_CMD_QUICK_COMMAND                0x00000027
#define I2C_MASTER_CMD_HS_MASTER_CODE_SEND          0x00000013
#define I2C_MASTER_CMD_FIFO_SINGLE_SEND             0x00000046
#define I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE          0x00000046
#define I2C_MASTER_CMD_FIFO_BURST_SEND_START        0x00000042
#define I2C_MASTER_CMD_FIFO_BURST_SEND_CONT         0x00000040
#define I2C_MASTER_CMD_FIFO_BURST_SEND_FINISH       0x00000044
#define I2C_MASTER_CMD_FIFO_BURST_SEND_ERROR_STOP   0x00000004
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_START     0x0000004a
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_CONT      0x00000048
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_FINISH    0x00000044
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_ERROR_STOP 0x00000004
```

➢ TM4C1294的I2C模块的使用

– 11. 查询I2CMCS的BUSBSY位，直到该位变为0

I2C 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	DATACK	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

```
#define I2C_MCS_BUSBSY     0x00000040    // Bus Busy
```

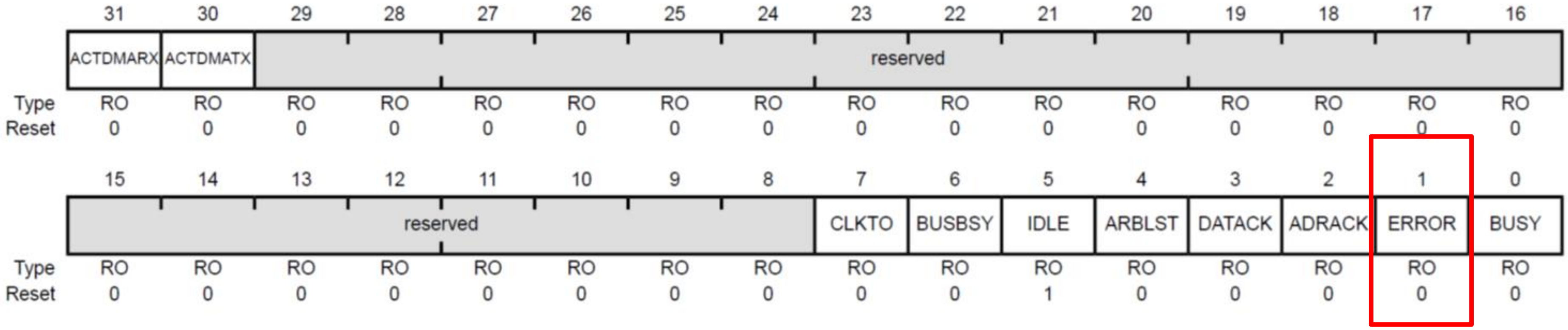
```
bool I2CMasterBusBusy(uint32_t ui32Base)
{
    // Check the arguments.
    ASSERT(_I2CBaseValid(ui32Base));
    // Return the bus busy status.
    if(HWREG(ui32Base + I2C_O_MCS) & I2C_MCS_BUSBSY)
    {         return(true);     }
    else {         return(false);     }
}
```

➢ TM4C1294的I2C模块的使用

– 12. 检查错误

I2C 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	DATACK	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。

I2C模块的使用

```
uint32_t
I2CMasterErr(uint32_t ui32Base)
{
    uint32_t ui32Err;
    // Check the arguments.
    ASSERT(_I2CBaseValid(ui32Base));
    // Get the raw error state
    ui32Err = HWREG(ui32Base + I2C_O_MCS);
    // If the I2C master is busy, then all the other bit are invalid, and
    // don't have an error to report.
    if(ui32Err & I2C_MCS_BUSY)
    {
        return(I2C_MASTER_ERR_NONE);
    }
    // Check for errors.
    if(ui32Err & (I2C_MCS_ERROR | I2C_MCS_ARBLST))
    {
        return(ui32Err & (I2C_MCS_ARBLST | I2C_MCS_DATACK | I2C_MCS_ADRACK));
    }
    else
    {
        return(I2C_MASTER_ERR_NONE);
    }
}
```

➢ TM4C1294的I2C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);
```

谢谢！