IIC 3—通信的应用

程晨闻
东南大学电气工程学院

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**
- 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

➢ 向从机0x1D写一个数据0x32，应该怎样操作寄存器?

1) 将从机地址0x1D写入**I2CMSA**， **R/S = 0**

（I2CMSA = 0x3A）

2) 将要发送的数据0x32写入 **I2CMDR（I2CMDR = 0x32）**

3) 检查总线是否忙碌

4) 将---**0-111**写入**I2CMCS（I2CMCS = 0x07）**

5) 等到控制器忙碌结束

6) 检查错误

➢ 从从机0x1D读一个数据，应该怎样操作寄存器?

1) 将从机地址写入I2CMSA， R/S = 1（I2CMSA = 0x3B）

2) 检查总线是否忙碌

3) 将---00111写入I2CMCS（I2CMCS = 0x07）

4) 等到控制器忙碌结束

5) 检查是否有错误

6) 从I2CMDR读出数据

内容概要

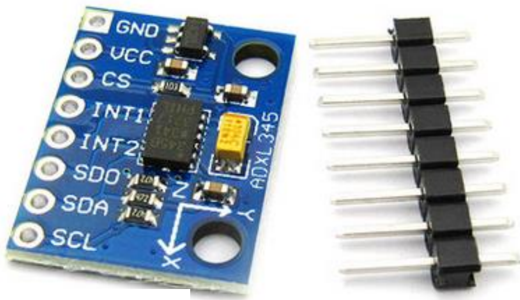
- TM4C1294的I2C模块的使用
- TM4C1294的I2C模块的应用

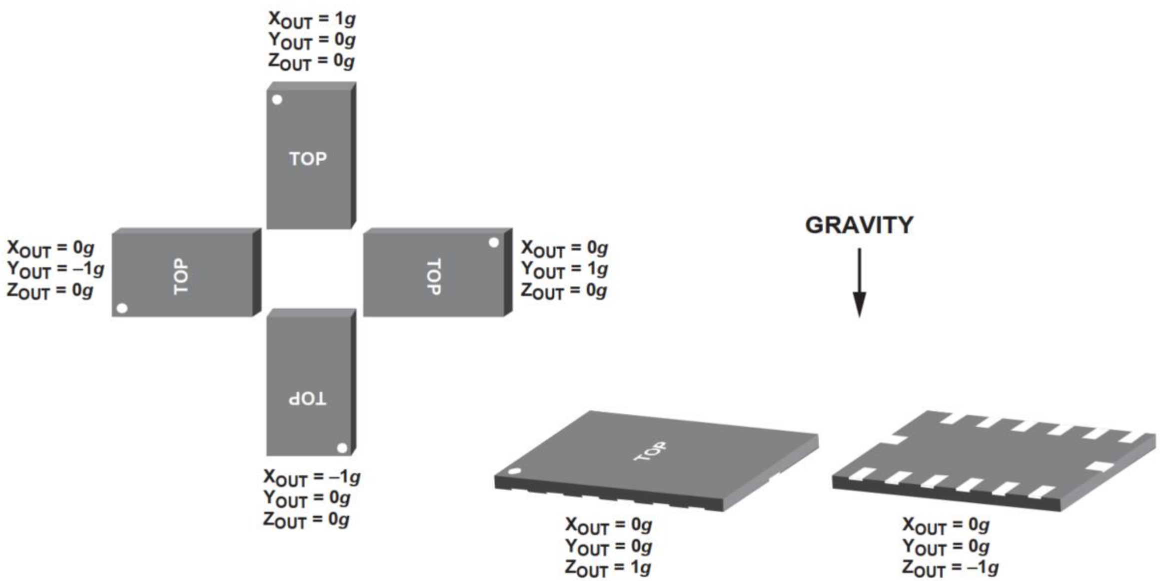
➢ TM4C1294的I2C模块的应用

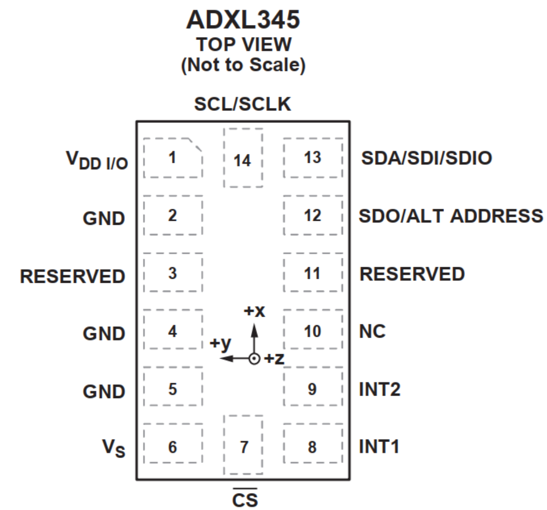
– ADXL345三轴加速度传感器

能够在预先不知道物体运动方向的场合下，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**；







TM4C1294的I2C模块的应用

– ADXL345三轴加速度传感器

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

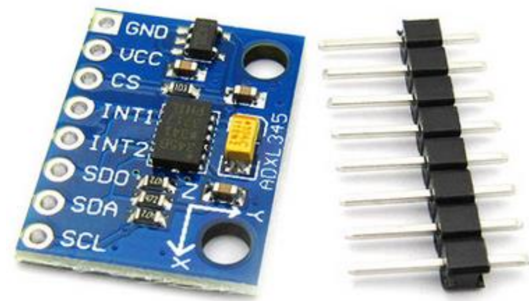
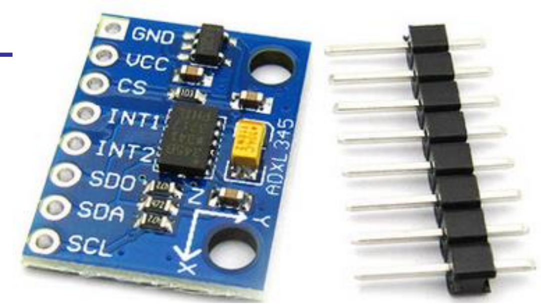


表5. 引脚功能描述

引脚编号	引脚名称	描述
1	$V_{DD\ I/O}$	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_S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_S	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^2C地址选择(I2C)
13	SDA/SDI/SDIO	串行数据(I^2C)/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^2C时钟，SCLK为SPI时钟。

➢ TM4C1294的I2C模块的应用

– ADXL345三轴加速度传感器



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

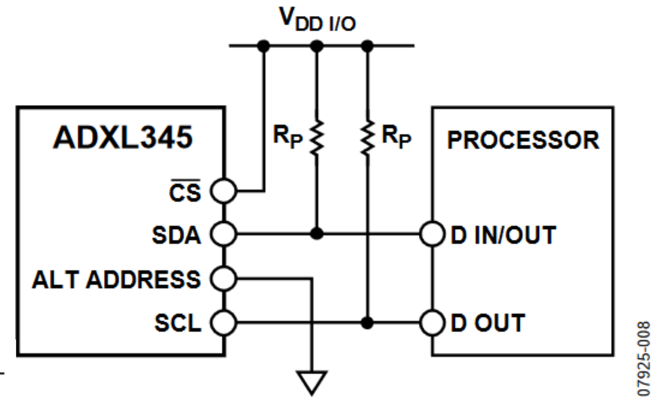


Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I²C Address Select (I²C).
13	SDA/SDI/SDIO	Serial Data (I²C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I²C, and SCLK is the clock for SPI.

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；
地址后接R / W位

将CS引脚**拉高**，ADXL345处于**I2C**模式

支持标准(**100 kHz**)和快速(**400 kHz**)数据传输模式

➢ TM4C1294的I2C模块的应用：

– ADXL345三轴加速度传感器

- ADXL345通过内部寄存器控制芯片并读写数据：

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVID	R	11100101	Device ID
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/W	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

TM4C1294的I2C模块的应用

– ADXL345三轴加速度传感器

- ADXL345通过内部寄存器控制芯片并读写数据：

1. DEVID 器件编号

Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5 (345 octal).

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正确，说明I2C通信能够正常工作。

– ADXL345三轴加速度传感器

- ADXL345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

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

Table 21. *g* Range Setting

Setting		
D1	D0	*g* Range
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

INT_INVERT Bit

A value of 0 in the INT_INVERT bit sets the interrupts to active high, and a value of 1 sets the interrupts to active low.

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor.

Justify Bit

A setting of 1 in the justify bit selects left-justified (MSB) mode, and a setting of 0 selects right-justified mode with sign extension.

Range Bits

These bits set the *g* range as described in Table 21.

➢ 为什么要看英文文档?

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 g range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

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。

– ADXL345三轴加速度传感器

- ADXL345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

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

– ADXL345三轴加速度传感器

- ADXL345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

Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)

These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

– ADXL345三轴加速度传感器

- ADXL345三轴加速度传感器的操作流程：

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**（DATAX0、DATAX1、DATAY0、DATAY1、DATAZ0和DATAZ1寄存器）

   读多个寄存器

– ADXL345三轴加速度传感器

- 通过I2C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

读单个寄存器：

SINGLE-BYTE READ											
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START[1]	SLAVE ADDRESS + READ			NACK	STOP
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);
```

– ADXL345三轴加速度传感器

- 通过I2C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（方法一）：

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);
```

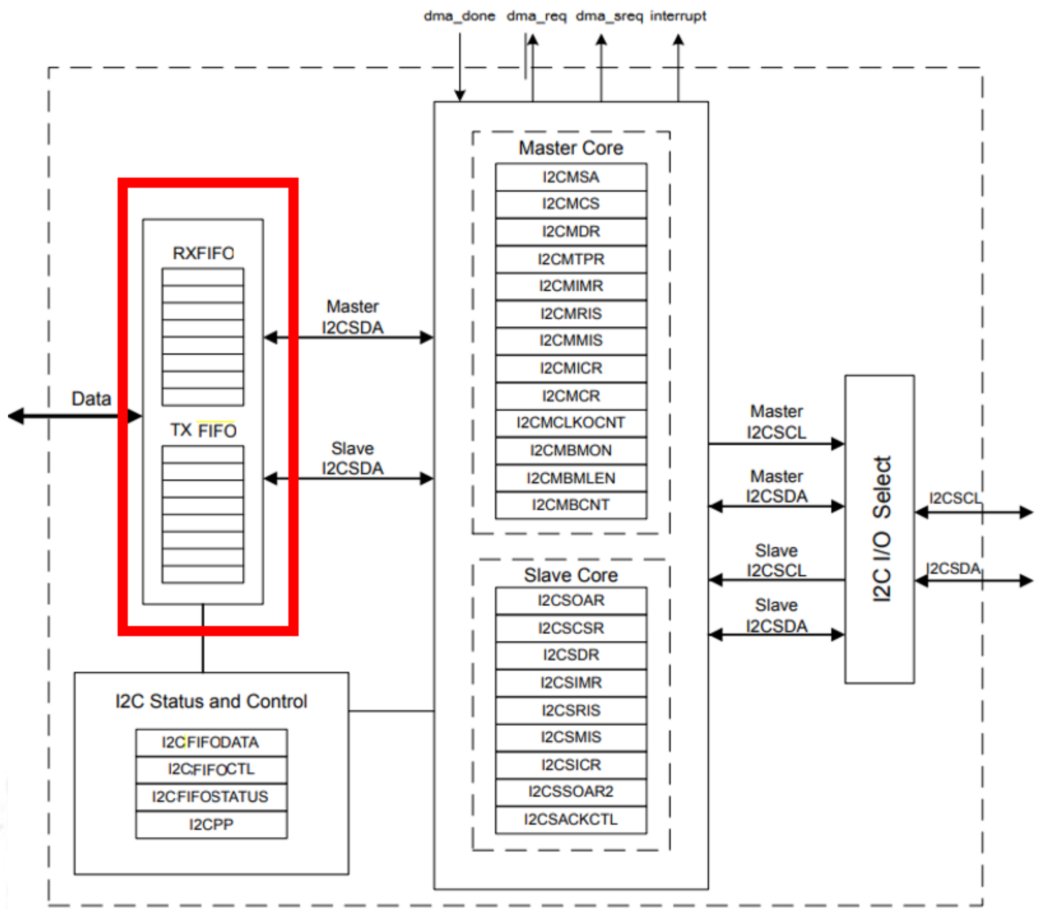
– ADXL345三轴加速度传感器

- 通过I2C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I2C模块的FIFO功能



– ADXL345三轴加速度传感器

- 通过I2C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I2C模块的FIFO功能

TM4C1294的I2C模块的FIFO和Burst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

– ADXL345三轴加速度传感器

- 通过I2C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
Type	WO	WO	WO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0

写单个寄存器（方法二）：使用I2C模块的FIFO功能

寄存器I2CFIFODATA

	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	I²C RX FIFO Read Data Byte This field contains the current byte being read in the RX FIFO stack.
7:0	DATA	WO	0	I²C TX FIFO Write Data Byte This field contains the current byte written to the TX FIFO. For back to back transmit operations, the application should not switch between writing to the **I2CSDR** register and the **I2CFIFODATA**.

– ADXL345三轴加速度传感器

- 通过I2C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I2C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	RO
Reset	0	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	RO
Reset	0	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

– ADXL345三轴加速度传感器

- 通过I2C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I2C模块的FIFO功能

使用**I2CFIFODataPut**函数，可以将数据填入**I2CFIFODATA**寄存器中

```
#define I2C_FIFOSTATUS_TXFF     0x00000002  // TX FIFO Full
```

```
void
I2CFIFODataPut(uint32_t ui32Base, uint8_t ui8Data)
{
    // Check the arguments.
    //
    ASSERT(_I2CBaseValid(ui32Base));
    // Wait until there is space.
    //
    while(HWREG(ui32Base + I2C_O_FIFOSTATUS) & I2C_FIFOSTATUS_TXFF)
    {
    }
    // Place data into the FIFO.
    //
    HWREG(ui32Base + I2C_O_FIFODATA) = ui8Data;
}
```

– ADXL345三轴加速度传感器

- 通过I2C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I2C模块的FIFO功能

使用**I2CFIFODataGet**函数，可以将数据从**I2CFIFODATA**寄存器中读出：

```
#define I2C_FIFOSTATUS_RXFE     0x00010000  // RX FIFO Empty
```

```
uint32_t
I2CFIFODataGet(uint32_t ui32Base)
{
    // Check the arguments.
    //
    ASSERT(_I2CBaseValid(ui32Base));
    // Wait until there is data to read.
    //
    while(HWREG(ui32Base + I2C_O_FIFOSTATUS) & I2C_FIFOSTATUS_RXFE)
    {
    }
    // Read a byte.
    //
    return(HWREG(ui32Base + I2C_O_FIFODATA));
}
```

– ADXL345三轴加速度传感器

- 通过I2C总线操作寄存器的方式

写单个寄存器（方法二）：使用I2C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。

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

– ADXL345三轴加速度传感器

- 通过I2C总线操作寄存器的方式

写单个寄存器（方法二）：使用I2C模块的FIFO功能

突发模式的传输数据的数量由I2CMBLEN寄存器决定

I2C Master Burst Length (I2CMBLEN)

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 0x030
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								CNTL							
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

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	CNTL	RW	0	I^{2}C Burst Length This field contains the programmed length of bytes of the Burst Transaction. If BURST is enabled this register must be set to a non-zero value otherwise an error will occur.

可以通过I2CMasterBurstLengthSet函数，设置I2CMBLEN寄存器

```
Void I2CMasterBurstLengthSet(uint32_t ui32Base, uint8_t ui8Length)
{
    …
    // Set the burst length.
    HWREG(ui32Base + I2C_O_MBLEN) = ui8Length;}
```

– ADXL345三轴加速度传感器

- 通过I2C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I2C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);
```

– ADXL345三轴加速度传感器

- 通过I2C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[1]	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DATAX0的地址0x32（**寄存器地址**）
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. …
8. 发送**停止位**

I2C模块的应用

读多个寄存器（方法一）：使用I2CMCS寄存器控制逐个读取

I2CMCS

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

MULTIPLE-BYTE READ												
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START[1]	SLAVE ADDRESS + READ		ACK		NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		DATA	

```
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); //0x03
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);
    }
```

– ADXL345三轴加速度传感器

- 通过I2C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（方法二）：使用FIFO连续读取数据

MULTIPLE-BYTE READ																	
MASTER	START		SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START[1]		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DATAX0的地址0x32（**寄存器地址**）
3. 发送**起始位**和**从机地址**，**读**
4. 连续读取多个数据，存储在**FIFO**中
5. 发送**停止**位
6. 将数据**从FIFO中读出**

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

读多个寄存器（方法二）：使用FIFO连续读取数据

MULTIPLE-BYTE READ

MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START	SLAVE ADDRESS + READ			ACK		NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		DATA		

```
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); //0x03
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++;}
```

– ADXL345三轴加速度传感器

将DATAX0、DATAX1两个八位的数据，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];
```

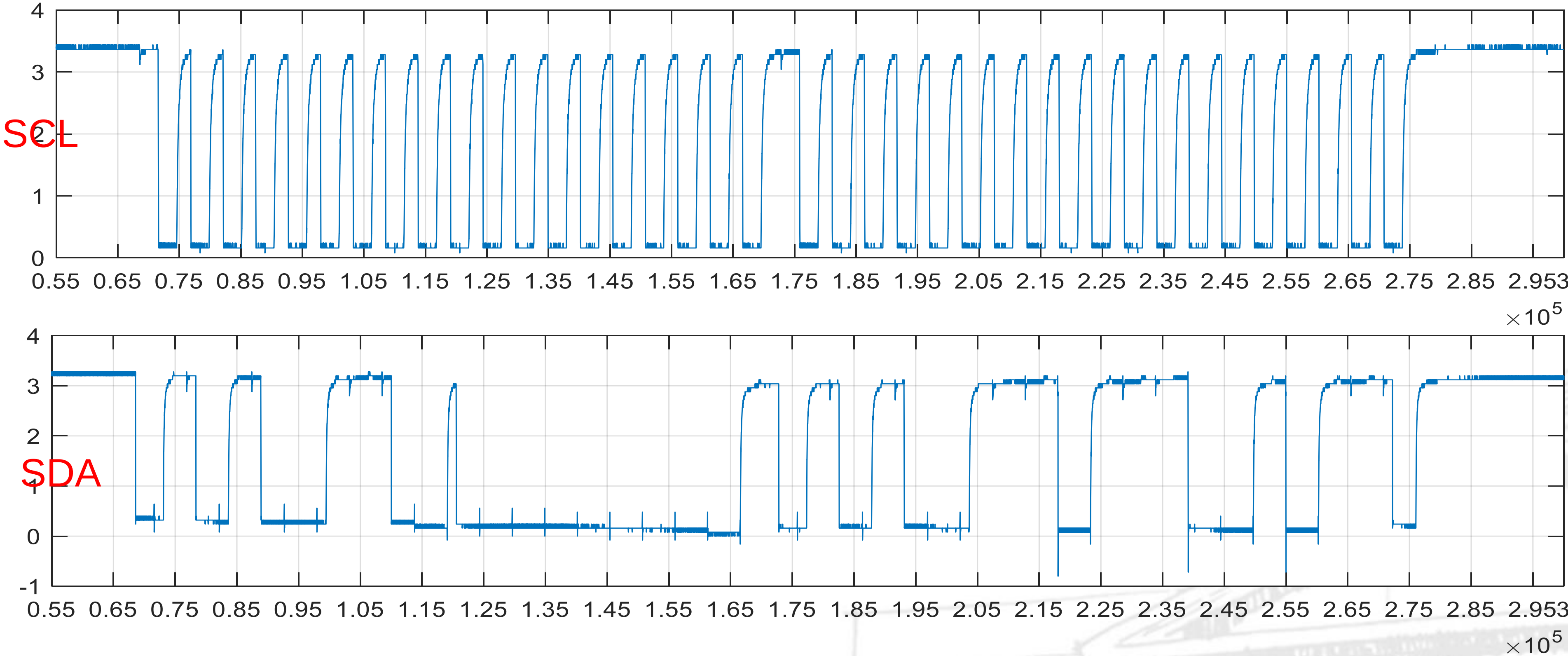
- 将这个16位的数据转换为加速度
- 根据重力加速度的分布，获知传感器的倾角状态

小结

- I2C总线的基本概念
- I2C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I2C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？



谢谢！