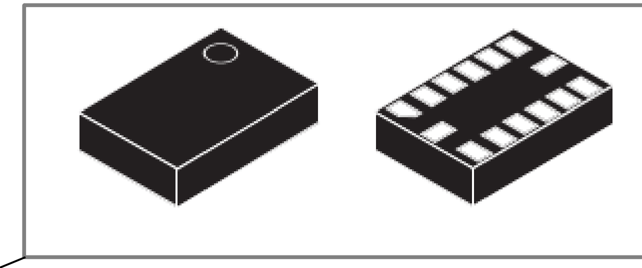
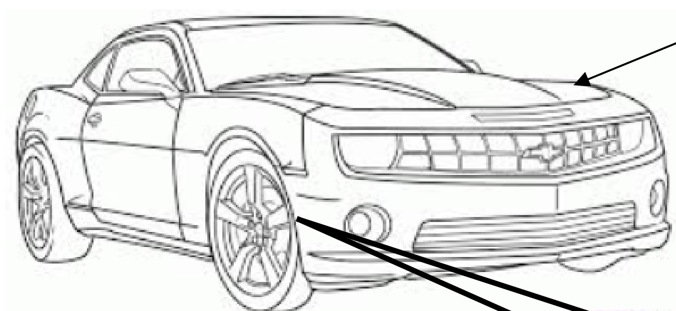
Sensors for Driving Direction and Turning Angle

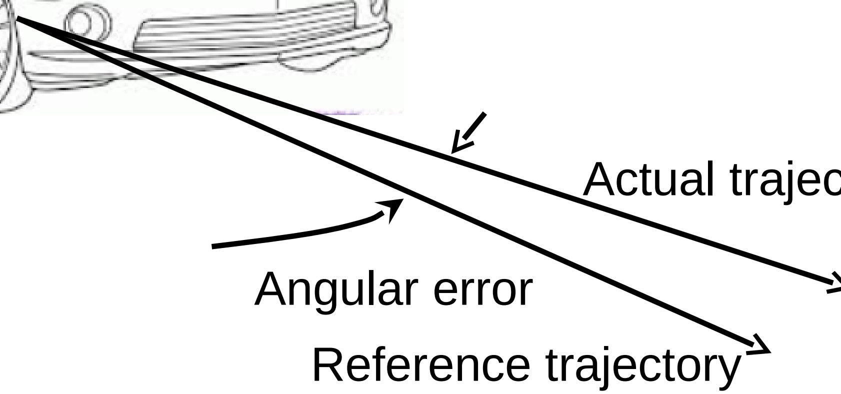
eCompass module:
3D accelerometer and 3D magnetometer

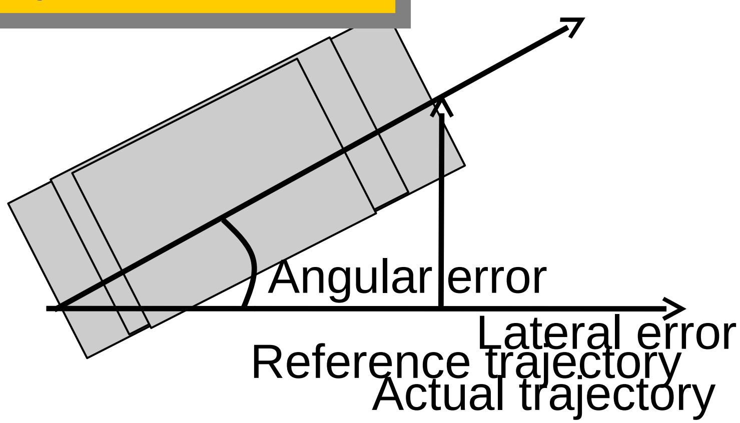


Caution: Steering sensor input is not necessarily the real angle of the vehicle, “skipping” may occur



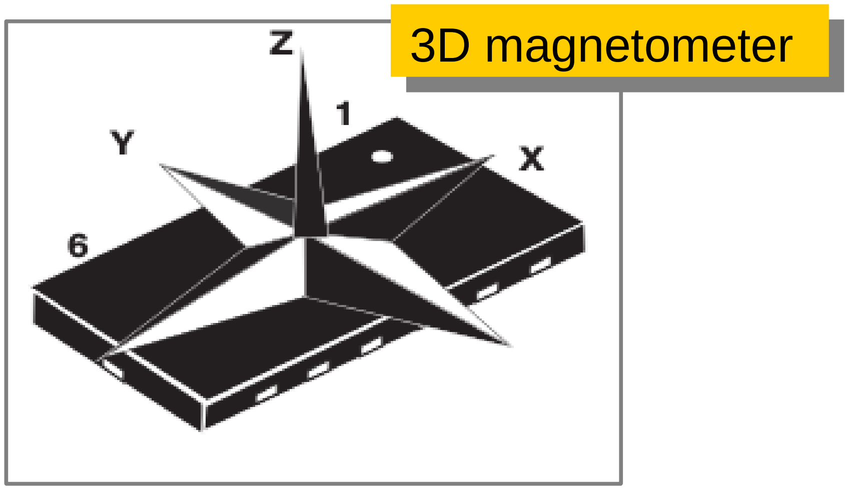
Use LSM303 or equivalent to sense the direction of the vehicle





The LSM303DLHC includes an I 2 C serial bus interface that supports standard and fast mode 100 kHz and 400 kHz. The system can be configured to generate interrupt signals by inertial wake-up/free-fall events as well as by the position of the device itself.

3D Accelerometer and 3D Magnetomete LMS303



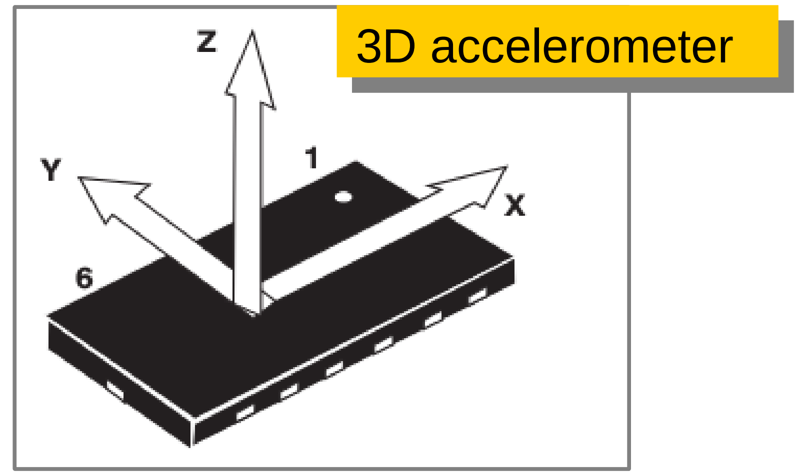


Table 9

Pin name	Pin description
SCL	I^2C serial clock (SCL)
SDA	I^2C serial data (SDA)

Reference: Table 9, pp 19, from LSM303 datasheet

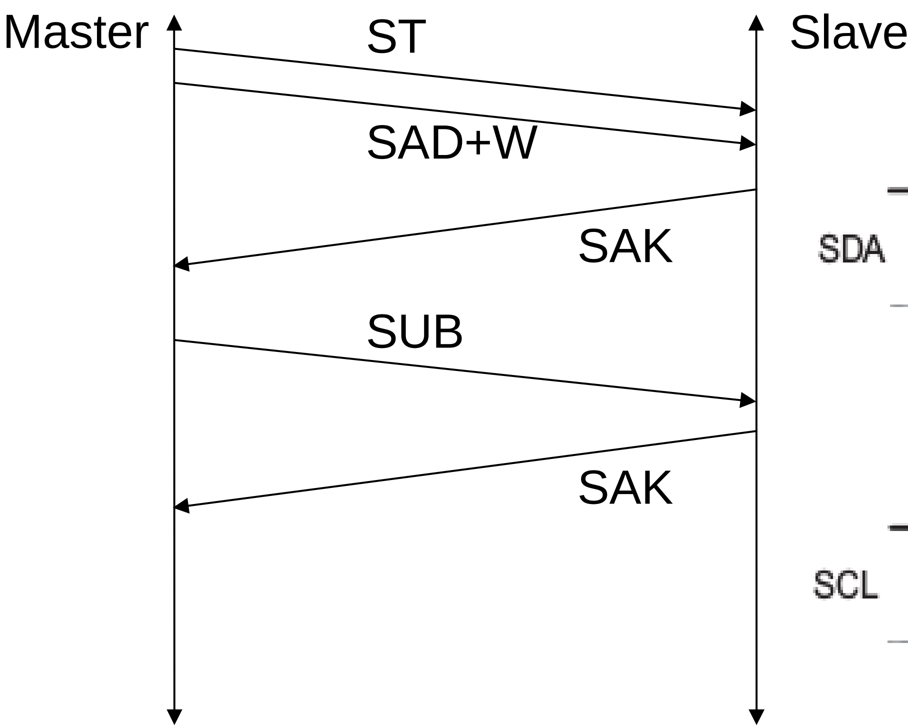
I2C Interface

(1) The transaction started through a START (ST) signal, defined as a high-to-low on the data line while the SCL line is held high.
(2) After ST, the next byte contains the slave address (the first 7 bit), bit 8 for if the master is receiving or transmitting data.
(3) When an address sent, each device compares the first seven bits after ST. If they match, the device is addressed.

I2C Handshaking LMS303

Table 11. Transfer when master is writing one byte to slave, pp 20, datasheet

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	



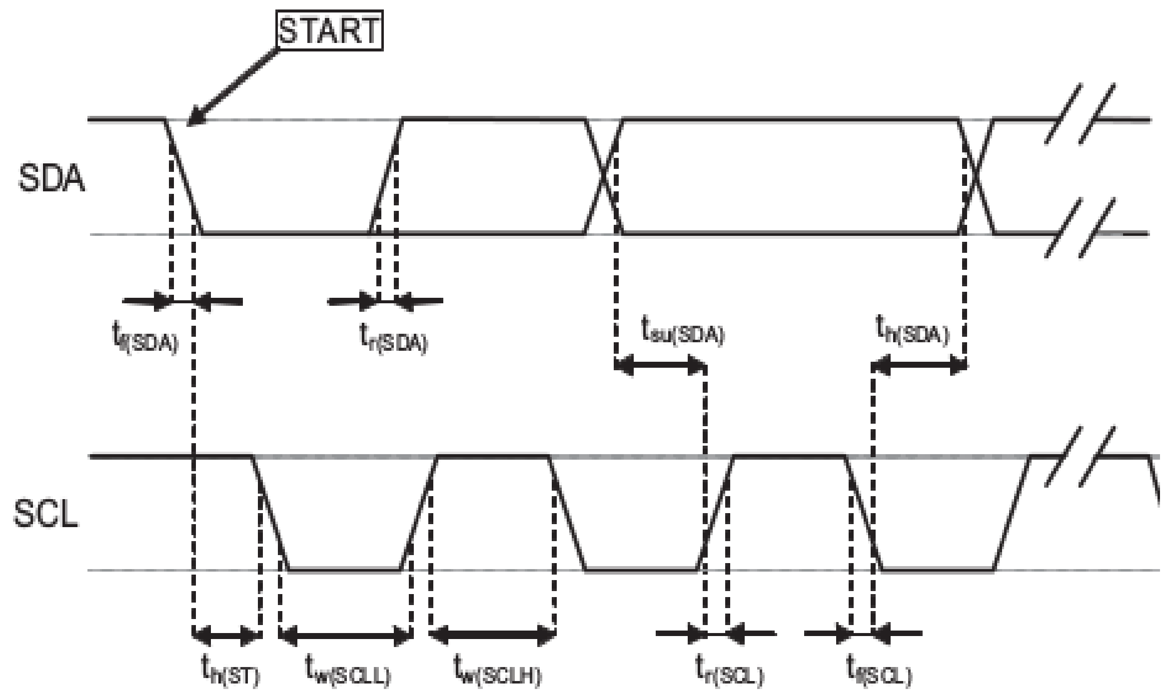
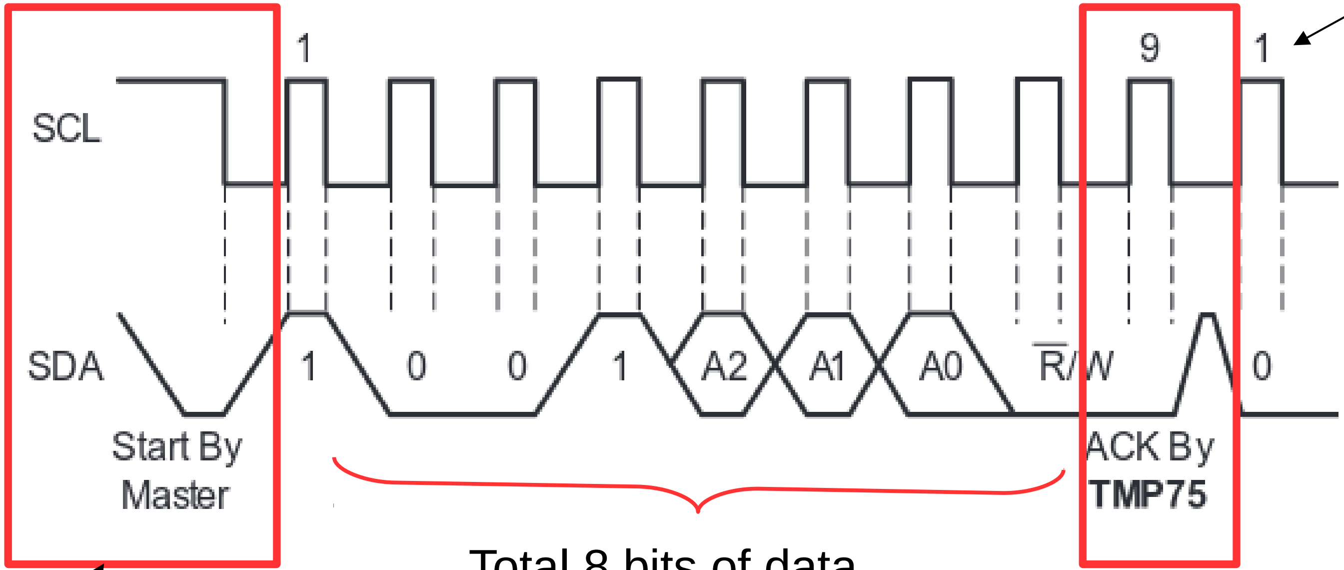


Figure 3. I 2 C slave timing diagram, pp 13 from datasheet

I2C Waveform Reference

Since LSM303 sensor datasheet did not provide the detailed waveform, we are using ti i2c temperature sensor as an example. The i2c waveform should be common to both devices

Starting the next frame



ST by master, e.g., sda goes low while scl holds high

Total 8 bits of data from the master

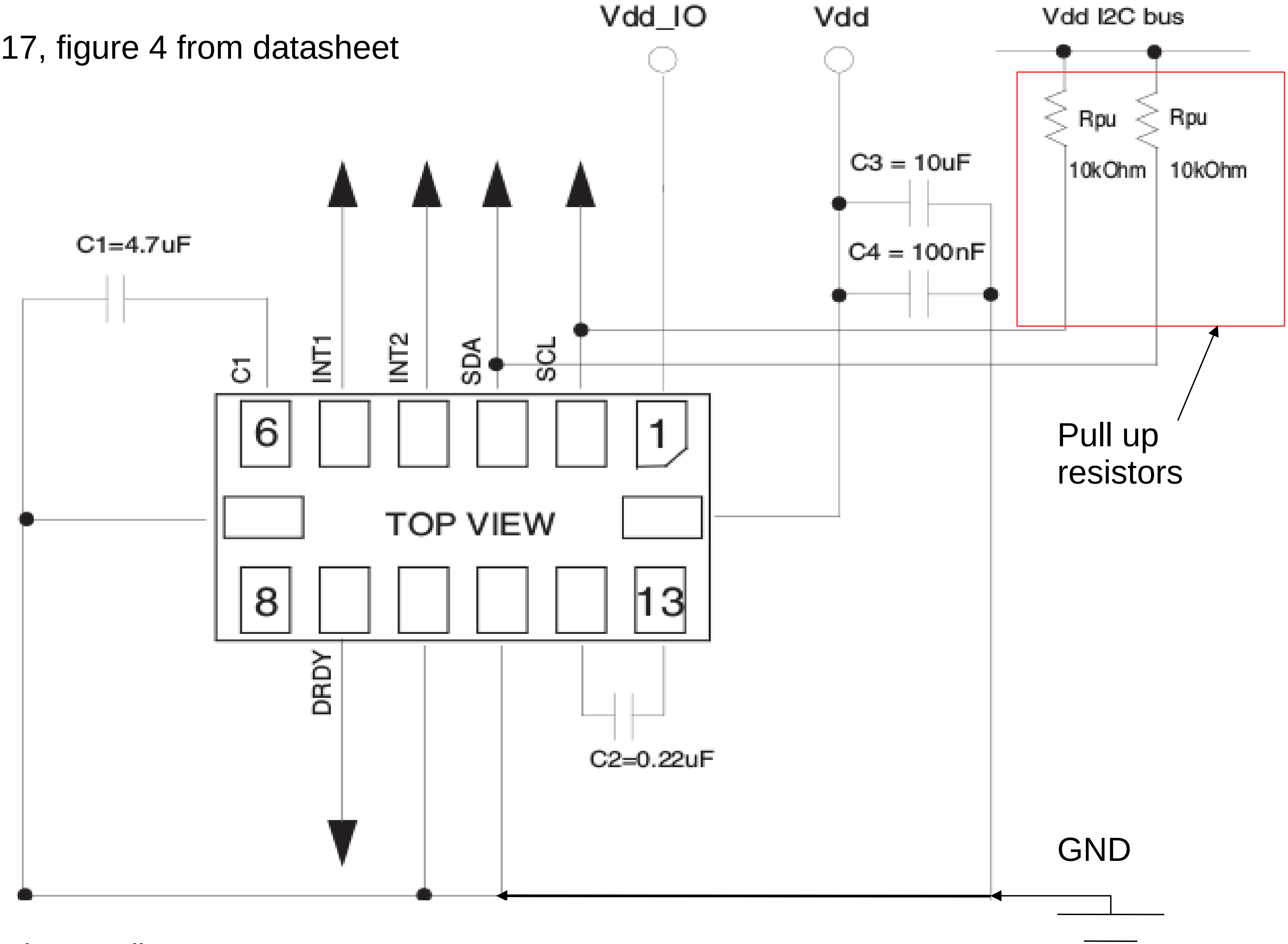
The SAK slave acknowledges the 8 bits of data from the master

http://www.ti.com/lit/ds/symlink/tmp175.pdf

Harry Li, Ph.D. April 2015

3D Accelerometer/Magnetomete Wire Diagram

pp. 17, figure 4 from datasheet



Harry Li, Ph.D. April 2015

I2C Sensor Init and Config

Table 11. Transfer when master is writing one byte to slave, pp 20, datasheet

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Table 12. Transfer when master is writing multiple bytes to slave, pp 20

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

1. Perform init and config by identify the i2c device (device address, from datasheet

2. identify the right sensor block with the sub-address, from datasheet

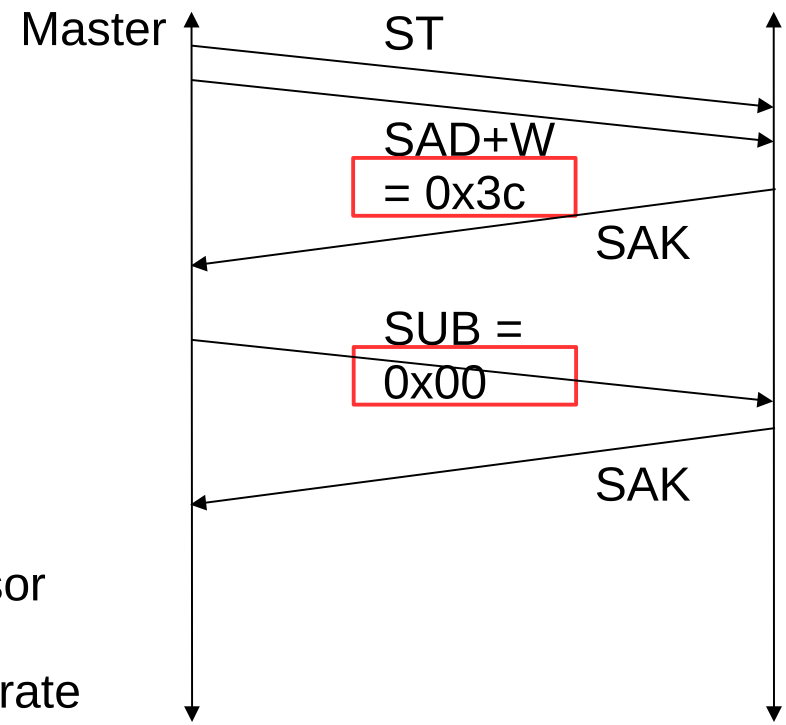
3. identify the control register(s) responsible for the init and config operations from datasheet

Steps for Init and Config (1)

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

1. Perform init and config by identify the i2c device (device address, from datasheet 0x3c, pp. 21)

For magnetic sensors the default (factory) 7-bit slave address is 0011110xb. The x bit is 0 for read and 1 for write



2. identify control register(s) for the right sensor block with the sub-address to set data rate
(1) CRA_REG_M register (0x00) to set data rate

TEMP_EN	0(1)	0(1)	DO2	DO1	DO0	0(1)	0(1)

Example; 3.0 Hz data rate, so 1 0 0 0 1 0 0 0 → 0x88, so write 0x88 to 0x00 location (CRA_REG_M)
DO2-DO1-DO0 = 0 1 0

Harry Li, Ph.D. Mar 2017

Steps for Init and Config (2)

		1		2		3				
Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

3. identify the control register(s) responsible for gain setting, from datasheet, pp 37, table 73, 75

Note; set smaller gain for stronger Gauss, adaptation algorithm may be needed

GN2	GN1	GN0	0(1)	0(1)	0(1)	0(1)	0(1)

Identify the control register CRB_REG_M (01h) for gain settings

GN2	GN1	GN0	Sensor input field range [Gauss]	Gain X, Y, and Z [LSB/Gauss]	Gain Z [LSB/Gauss]	Output range
0	0	1	±1.3	1100	980	
0	1	0	±1.9	855	760	
0	1	1	±2.5	670	600	
1	0	0	±4.0	450	400	0xF800–0x07FF (-2048–2047)

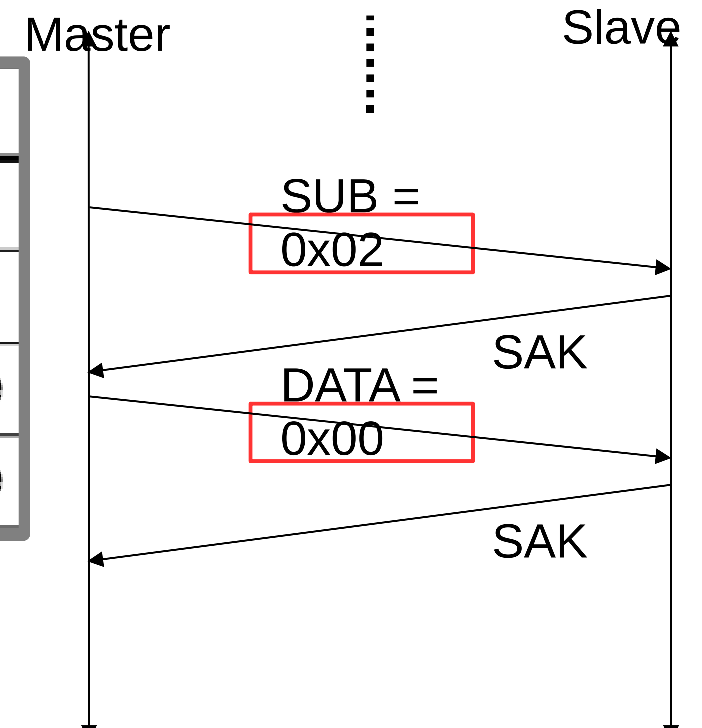
Steps for Init and Config (3)

Master	ST	SAD + W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

4. identify the control register responsible for mode of operation, from datasheet, pp 37, table 76, 78   MR_REG_M (02h)

$0^{(1)}$	$0^{(1)}$	$0^{(1)}$	$0^{(1)}$	$0^{(1)}$	$0^{(1)}$	MD1	MD0

MD1	MD0	Mode
0	0	Continuous-conversion mode
0	1	Single-conversion mode
1	0	Sleep-mode. Device is placed in sleep-mode
1	1	Sleep-mode. Device is placed in sleep-mode



Select continuous conversion mode, so set the register to 00, e.g., write 0x00 to address 0x02

Harry Li, Ph.D. Mar 2017

Read the Sensor Data

Master	ST	SAD +W		SUB		SR	SAD +R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Repeated start

Master Acknowledgment

No Master Acknowledgment

5. Identify the data register(s) to read sensor data back to a buffer of the user application program, from datasheet, pp 38

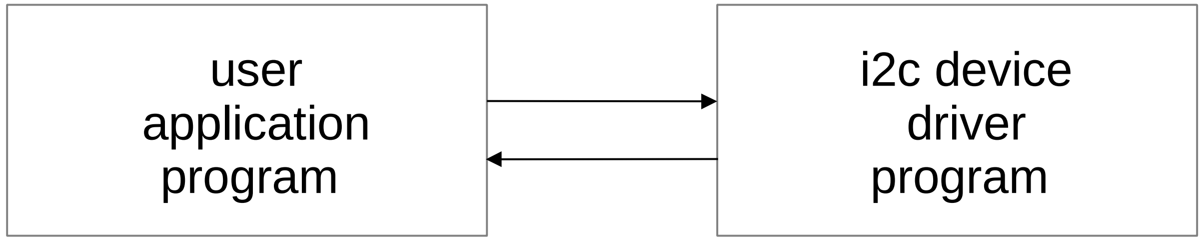
OUT_X_H_M (03), OUT_X_L_M (04h)
OUT_Z_H_M (05), OUT_Z_L_M (06h)
OUT_Y_H_M (07), OUT_Y_L_M (08h)

The data is in 2's complement form.

Therefore, to read the sensor date, we will have to assign the content at addresses 0x03, 0x04, …, 0x08 to 6 bytes buffers, the first 2 byte for X and the next 2 bytes for Z, and the final 2 bytes for Y.

Harry Li, Ph.D. Mar 2017

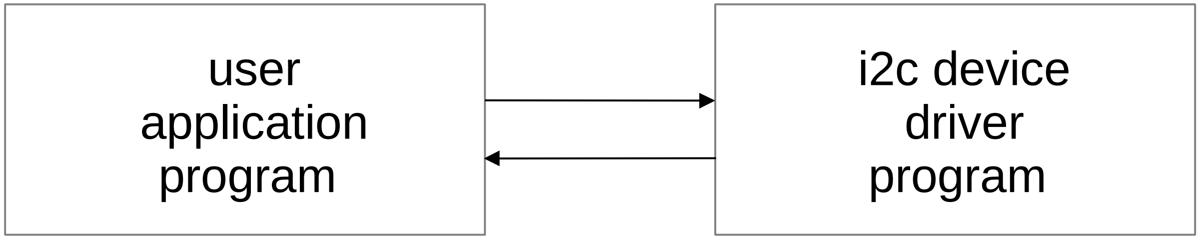
C Code for the Init and Config 1



Sample code from /mini6410/linux/examples/eeprog.c

```
int main(int argc, char** argv)
{
        struct eeprom e;
        int op;    op = 0;
        usage_if(argc != 2 || argv[1][0] != '-' || argv[1][2] != '\0');
        op = argv[1][1];
        fprintf(stderr, "Open /dev/i2c/0 with 8bit mode\n");
        die_if(eeprom_open("/dev/i2c/0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e) < 0,
                "unable to open eeprom device file "
                "(check that the file exists and that it's readable)");
        switch(op)
        {
        case 'r':  fprintf(stderr, "  Reading 256 bytes from 0x0\n");
            read_from_eeprom(&e, 0, 256);
            break;
        case 'w':  fprintf(stderr, "  Writing 0x00-0xff into 24C08 \n");
            write_to_eeprom(&e, 0);
            break;
```

C Code for the Init and Config 2




```
int eeprom_open(char *dev_fqn, int addr, int type, struct eeprom* e)
{
        int funcs, fd, r;
        e->fd = e->addr = 0;
        e->dev = 0;
        fd = open(dev_fqn, O_RDWR);
        if(fd <= 0)
        { fprintf(stderr, "Error eeprom_open: %s\n", strerror(errno));
                return -1;
        }
        // get funcs list
        if((r = ioctl(fd, I2C_FUNCS, &funcs) < 0))
        {
                fprintf(stderr, "Error eeprom_open: %s\n", strerror(errno));
                return -1;
        }
```

C Code for the Init and Config 3

i2c device
driver
program

From Kconfig file below, driver source code in /drive

```
config I2C_CHARDEV
        tristate "I2C device interface"
        help
          Say Y here to use i2c-* device files, usually found in the /dev
          directory on your system.  They make it possible to have user-space
          programs use the I2C bus.  Information on how to do this is
          contained in the file <file:Documentation/i2c/dev-interface>.

          This support is also available as a module.  If so, the module
          will be called i2c-dev.
```

Driver source
code in
/drivers/i2c