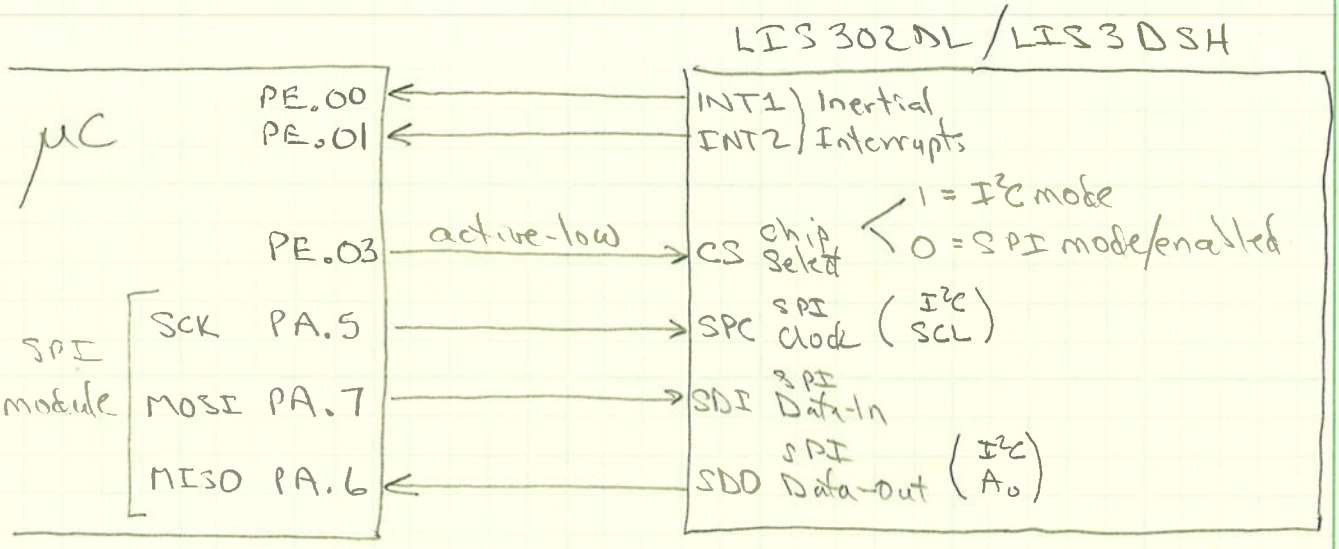
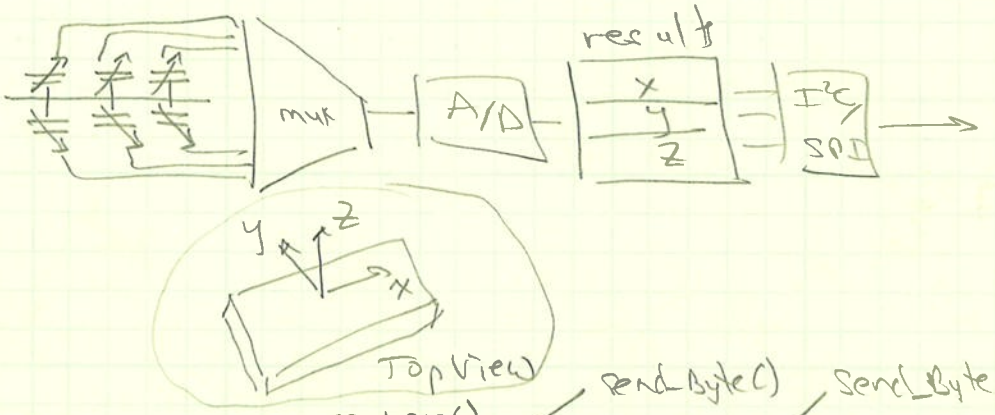
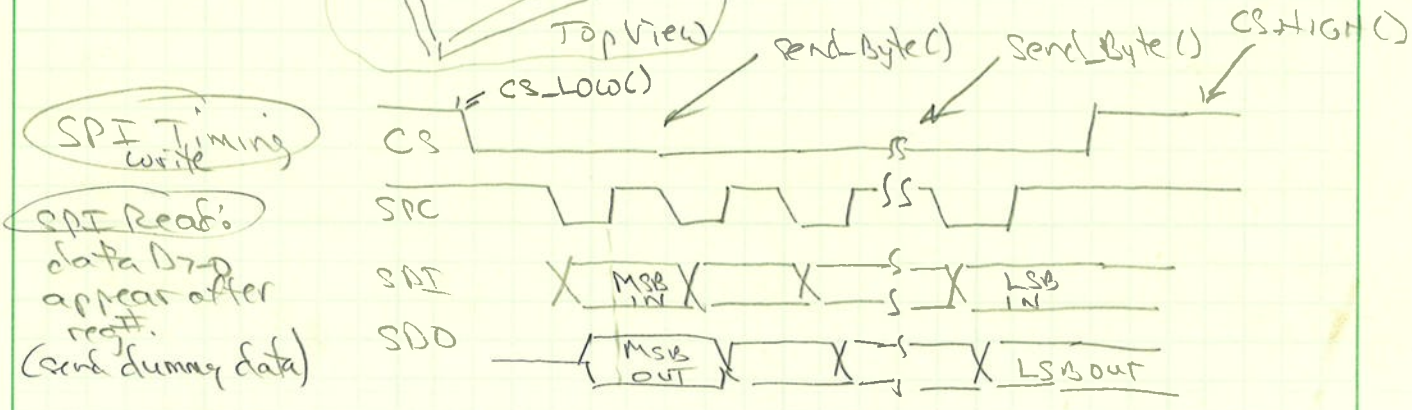
Accelerometer | STM32F4-Discovery Board



4-wire SPI: CS, SPC, SDI, SDO

3-wire SPI: CS, SPC, SDI/SDO (on SDI pin)
($\frac{1}{2}$ duplex)





SDI: (16 bits) R/$\overline{W}$ MS $AD_5 .. AD_0$ $DI_7 .. DI_0$

R/$\overline{W}$: 1 = read, 0 = write

MS: 1 = don't increment address; 0 = incr. address & send more data

$AD_5 .. AD_0$: reg. #

$DI_7 .. DI_0$: data

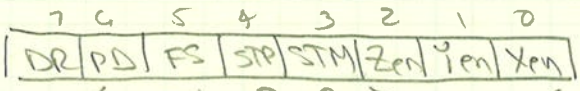
REGISTERS - next page

OUT_X (29) } 8-bit accel. data
OUT_Y (2B) }
OUT_Z (2D) }

STATUS_REG (27) - new data available/overrun errors
CTRL_REG1 - enable axes/data rate/scale/pwr
CTRL_REG2 - serial interface mode/reboot/high-pass filter
CTRL_REG3 - interrupt ctrl.

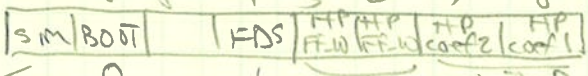
LIS302DL

CTRL_REG1



- DR — Data Rate: 0 = 100Hz, 1 = 400Hz
- PD — Power Down: 0 = down, 1 = active
- STP / STM — 0 0 = no self-test
- Zen, Yen, Xen — 1 to enable Z, Y, X axes
- FS — Full scale selection: 0 = ± [2 - 2.3] g; 1 = ± [8 - 9.2] g

CTRL_REG2



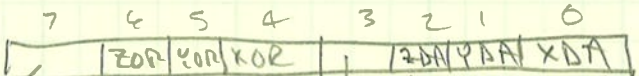
- SIM — SPI Serial Interface Mode: 0: 4-wire, 1: 3-wire
- BOOT — 0: normal mode
- FDS — High-pass Filter: 0 = bypass, 1 = enable
- HP FF-WU2 / HP FF-WU1 — High Pass Filter Enable for wake-up
- HP coef2 / HP coef1 — High Pass Filter Cut-off Freq.

	@100Hz	@400Hz
00	2 Hz	8 Hz
01	1	4
10	0.5	2
11	0.25	1

CTRL_REG3

- IHL — 0 = active high, 1 = active low interrupt levels
- PP_OD — 0 = push-pull, 1 = open drain output
- I2CFG2-0 — Control INT2 pin
- I1CFG2-0 — Control INT1 pin

STATUS_REG



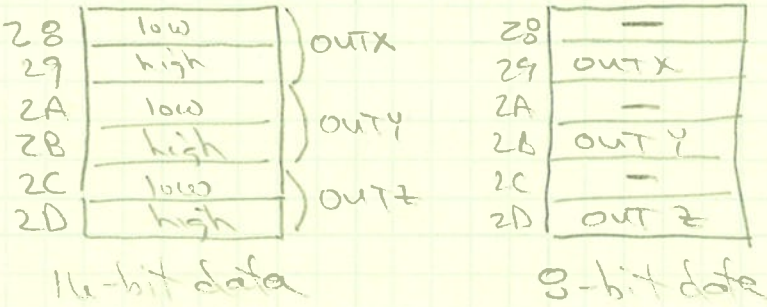
- ZYXOR — set of data overrun.
- Z/Y/X — data overrun
- ZYXDA — complete set of data available
- Z/Y/X Data available

OUT_X (29h)
OUT_Y (2Bh)
OUT_Z (2Dh)
} (8-bit) data for axes.

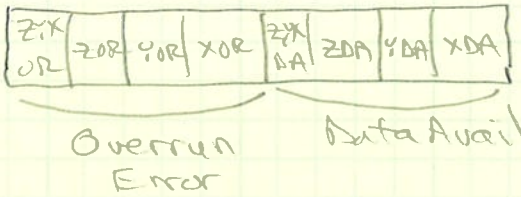
–	28
OUT_X	29
–	2A
OUT_Y	2B
–	2C
OUT_Z	2D

LIS3DSH Registers (vs. LI3302DL)

Output Data



Status:



27 - both chips

Control Registers

LI3302DL: CR1 (20) - data rate, power, enable x-y-z
scale (±2g, ±8g)

CR2 (21) - SPI interface (4 vs. 3-wire)
high-pass filter

CR3 (22) - interrupt pins
enable & configure

LIS3DSH:

CR4 (20) - more data rates, enable x-y-z
ODR_{3-0} : BDU : ZEN/YEN/XEN
(3.125Hz to 1600Hz) 0 = continuous update

CR3 (23) - set up interrupt pins: INT1 / INT2

CR5 (24) - BW_{2-1} : $FSCALE_{2-0}$: ST_{2-1} : SIM

Anti-alias filter BW	full scale select	self test	0 for 4-wire SPI
	±2g	00	
800Hz	4g		
400	6g		
200	8g		
50	16g		

CR6 (25) - FIFO control

Files to be added to the project

stm32f4_discovery_accelerometer.c

```
/* Includes Accelerometer component drivers and selects one after reading chip ID */
#include "..\Components\lis302dl\lis302dl.h"
#include "..\Components\lis3dsh\lis3dsh.h"

/* Accelerometer functions */
uint8_t BSP_ACCELERO_Init(void);       -- calls Init functions for lis302dl or lis3dsh accelerometer
uint8_t BSP_ACCELERO_ReadID(void); -- returns ID# of the chip
void    BSP_ACCELERO_Reset(void);
void    BSP_ACCELERO_GetXYZ(int16_t *pDataXYZ);  -- read and return accelerometer values
```

stm32f4_discovery.c

```
static void    SPIx_Init(void);                    // calls stm32f4xx_hal_spi.c  functions
static void    SPIx_MspInit(void);
static uint8_t  SPIx_WriteRead(uint8_t Byte);
static  void   SPIx_Error(void);

void        ACCELERO_IO_Init(void);   //Configures SPI interface to accelerometer
void        ACCELERO_IO_Write(uint8_t *pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite);
void        ACCELERO_IO_Read(uint8_t *pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead);

#define ACCELERO_CS_LOW()     HAL_GPIO_WritePin(ACCELERO_CS_GPIO_PORT, ACCELERO_CS_PIN,
GPIO_PIN_RESET)
#define ACCELERO_CS_HIGH()    HAL_GPIO_WritePin(ACCELERO_CS_GPIO_PORT, ACCELERO_CS_PIN,
GPIO_PIN_SET)
```

lis302dl.c (change LIS302DL to LIS3DSH in lis3dsh.c) - both needed if using discovery_accelerometer file

```
void   LIS302DL_Init(uint16_t InitStruct);
uint8_t LIS302DL_ReadID(void);
void   LIS302DL_FilterConfig(uint8_t FilterStruct);
void   LIS302DL_InterruptConfig(LIS302DL_InterruptConfigTypeDef *LIS302DL_IntConfigStruct);
void   LIS302DL_Click_IntConfig(void);
void   LIS302DL_Click_IntClear(void);
void   LIS302DL_LowpowerCmd(uint8_t LowPowerMode);
void   LIS302DL_FullScaleCmd(uint8_t FS_value);
void   LIS302DL_DataRateCmd(uint8_t DataRateValue);
void   LIS302DL_RebootCmd(void);
void   LIS302DL_ReadACC(int16_t *pData);
```

```
 96  /**
 97    * @brief  Setx Accelerometer Initialization.
 98    * @param  None
 99    * @retval ACCELERO_OK if no problem during initialization
100    */
101  uint8_t BSP_ACCELERO_Init(void)
102  {
103    uint8_t ret = ACCELERO_ERROR;
104    uint16_t ctrl = 0x0000;
105    LIS302DL_InitTypeDef         lis302dl_initstruct;
106    LIS302DL_FilterConfigTypeDef lis302dl_filter = {0,0,0};
107    LIS3DSH_InitTypeDef          l1s3dsh_InitStruct;
108
109    if(Lis302dlDrv.ReadID() == I_AM_LIS302DL)        - Initialize LIS302DL ?
110    {
111      /* Initialize the accelerometer driver structure */
112      AcceleroDrv = &Lis302dlDrv;
113
114      /* Set configuration of LIS302DL MEMS Accelerometer *********************/
115      lis302dl_initstruct.Power_Mode = LIS302DL_LOWPOWERMODE_ACTIVE;
116      lis302dl_initstruct.Output_DataRate = LIS302DL_DATARATE_100;
117      lis302dl_initstruct.Axes_Enable = LIS302DL_XYZ_ENABLE;
118      lis302dl_initstruct.Full_Scale = LIS302DL_FULLSCALE_2_3;
119      lis302dl_initstruct.Self_Test = LIS302DL_SELFTEST_NORMAL;
120
121      /* Configure MEMS: data rate, power mode, full scale, self test and axes */
122      ctrl = (uint16_t) (lis302dl_initstruct.Output_DataRate | lis302dl_initstruct.Power_Mode | \
123                         lis302dl_initstruct.Full_Scale | lis302dl_initstruct.Self_Test | \
124                         lis302dl_initstruct.Axes_Enable);
125
126      /* Configure the accelerometer main parameters */
127      AcceleroDrv->Init(ctrl);    calls LIS302DL_Init(ctrl)
128
129      /* MEMS High Pass Filter configuration */
130      lis302dl_filter.HighPassFilter_Data_Selection = LIS302DL_FILTEREDDATASELECTION_OUTPUTREGISTER;
131      lis302dl_filter.HighPassFilter_CutOff_Frequency = LIS302DL_HIGHPASSFILTER_LEVEL_1;
132      lis302dl_filter.HighPassFilter_Interrupt = LIS302DL_HIGHPASSFILTERINTERRUPT_1_2;
133
134      /* Configure MEMS high pass filter cut-off level, interrupt and data selection bits
    */
135      ctrl = (uint8_t)(lis302dl_filter.HighPassFilter_Data_Selection | \
136                       lis302dl_filter.HighPassFilter_CutOff_Frequency | \
137                       lis302dl_filter.HighPassFilter_Interrupt);
138
139      /* Configure the accelerometer LPF main parameters */
140      AcceleroDrv->FilterConfig(ctrl);    calls LIS302DL_FilterConfig
141
142      ret = ACCELERO_OK;
143    }
144    else if(Lis3dshDrv.ReadID() == I_AM_LIS3DSH)        - Initialize LIS3DSH ?
145    {
146      /* Initialize the accelerometer driver structure */
147      AcceleroDrv = &Lis3dshDrv;
148
149      /* Set configuration of LIS3DSH MEMS Accelerometer **********************/
150      l1s3dsh_InitStruct.Output_DataRate = LIS3DSH_DATARATE_100;
151      l1s3dsh_InitStruct.Axes_Enable = LIS3DSH_XYZ_ENABLE;
152      l1s3dsh_InitStruct.SPI_Wire = LIS3DSH_SERIALINTERFACE_4WIRE;
153      l1s3dsh_InitStruct.Self_Test = LIS3DSH_SELFTEST_NORMAL;
154      l1s3dsh_InitStruct.Full_Scale = LIS3DSH_FULLSCALE_2;
155      l1s3dsh_InitStruct.Filter_BW = LIS3DSH_FILTER_BW_800;
156
157      /* Configure MEMS: power mode(ODR) and axes enable */
158      ctrl = (uint16_t) (l1s3dsh_InitStruct.Output_DataRate | \
159                         l1s3dsh_InitStruct.Axes_Enable);
160
161      /* Configure MEMS: full scale and self test */
162      ctrl |= (uint16_t) ((l1s3dsh_InitStruct.SPI_Wire    | \
163                           l1s3dsh_InitStruct.Self_Test   | \
164                           l1s3dsh_InitStruct.Full_Scale  | \
165                           l1s3dsh_InitStruct.Filter_BW) << 8);
```

```
166
167       /* Configure the accelerometer main parameters */
168       AcceleroDrv->Init(ctrl);
169
170       ret = ACCELERO_OK;
171     }
172
173     else
174     {
175       ret = ACCELERO_ERROR;
176     }
177     return ret;
178   }
179
```

```
116  /**
117    * @brief  Set LIS302DL Initialization.
118    * @param  InitStruct: contains mask of different init parameters
119    * @retval None
120    */
121  void LIS302DL_Init(uint16_t InitStruct)
122  {
123    uint8_t ctrl = 0x00;
124
125    /* Configure the low level interface */
126    ACCELERO_IO_Init();
127
128    ctrl = (uint8_t) InitStruct;
129
130    /* Write value to MEMS CTRL_REG1 register */
131    ACCELERO_IO_Write(&ctrl, LIS302DL_CTRL_REG1_ADDR, 1);
132  }
133
```

- bit for CR1
- configure pins + SPI module
- CR1

CR1

7	6	5	4	3	2	1	0
DR	PD	FS	0	0	Zen	Yen	Xen

- DR: data rate — 0 = 100 Hz, 1 = 400 Hz
- PD: 0 pwr dn, 1 pwr on
- FS: full-scale — 0 = ±[2-23]g, 1 = ±[8-9.2]g
- Zen, Yen, Xen: enable Y, Y, Z axes

```
/**
  * @brief  Configures the Accelerometer SPI interface.
  * @param  None
  * @retval None
  */
void ACCELERO_IO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Configure the Accelerometer Control pins ----------------------------*/
  /* Enable CS GPIO clock and configure GPIO pin for Accelerometer Chip select */
  ACCELERO_CS_GPIO_CLK_ENABLE();

  /* Configure GPIO PIN for LIS Chip select */
  GPIO_InitStructure.Pin = ACCELERO_CS_PIN;
  GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStructure.Pull  = GPIO_NOPULL;
  GPIO_InitStructure.Speed = GPIO_SPEED_MEDIUM;
  HAL_GPIO_Init(ACCELERO_CS_GPIO_PORT, &GPIO_InitStructure);

  /* Deselect: Chip Select high */
  ACCELERO_CS_HIGH();

  SPIx_Init();
}
```

- PE_3 (not part of SPI module)

- Initialize SPI pins and module.
  PA_5 = SCK
  PA_7 = MOSI
  PA_6 = MISO

ACCELERO_* functions in this file.

```
351  /**
352    * @brief  Read LIS302DL output register, and calculate the acceleration
353    *         ACC[mg]=SENSITIVITY* (out_h*256+out_l)/16 (12 bit rappresentation)
354    * @param  pfData: Data out pointer
355    * @retval None
356    */
357  void LIS302DL_ReadACC(int16_t *pData)
358  {
359    int8_t buffer[6];
360    int16_t pnRawData[3];
361    uint8_t sensitivity = LIS302DL_SENSITIVITY_2_3G;
362    uint8_t crtl, i = 0x00;
363
364    ACCELERO_IO_Read(&crtl, LIS302DL_CTRL_REG1_ADDR, 1);
365    ACCELERO_IO_Read((uint8_t*)buffer, LIS302DL_OUT_X_ADDR, 6);
366
367    for(i=0; i<3; i++)
368    {
369      pnRawData[i] = buffer[2*i];
370    }
371
372    switch(crtl & LIS302DL_FULLSCALE_9_2)
373    {
374      /* FS bit = 0 ==> Sensitivity typical value = 18milligals/digit*/
375    case LIS302DL_FULLSCALE_2_3:
376      sensitivity = LIS302DL_SENSITIVITY_2_3G;
377      break;
378
379      /* FS bit = 1 ==> Sensitivity typical value = 72milligals/digit*/
380    case LIS302DL_FULLSCALE_9_2:
381      sensitivity = LIS302DL_SENSITIVITY_9_2G;
382      break;
383
384    default:
385      break;
386    }
387
388    /* Obtain the mg value for the three axis */
389    for(i=0; i<3; i++)
390    {
391      pData[i]=(pnRawData[i] * sensitivity);
392    }
393  }
394
```

return data

Similar function for LIS3DSH (but 16-bit data)

read/write register # (LIS302DL or LIS3DSH)

read 6 bytes (X, Y, Z)

data in alternate bytes

- undefined for LIS302DL

scale data by "sensitivity"

2 ranges of values:

± [2 - 2.3] g

± [8 - 9.2] g

```
/**
 * @brief Reads a block of data from the Accelerometer.
 * @param pBuffer: pointer to the buffer that receives the data read from the Accelerometer.
 * @param ReadAddr: Accelerometer's internal address to read from.
 * @param NumByteToRead: number of bytes to read from the Accelerometer.
 * @retval None
 */
void ACCELERO_IO_Read(uint8_t *pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead)
{
 if(NumByteToRead > 0x01)
 {
  ReadAddr |= (uint8_t)(READWRITE_CMD | MULTIPLEBYTE_CMD);
 }
 else
 {
  ReadAddr |= (uint8_t)READWRITE_CMD;
 }
 /* Set chip select Low at the start of the transmission */
 ACCELERO_CS_LOW();

 /* Send the Address of the indexed register */
 SPIx_WriteRead(ReadAddr);

 /* Receive the data that will be read from the device (MSB First) */
 while(NumByteToRead > 0x00)
 {
  /* Send dummy byte (0x00) to generate the SPI clock to ACCELEROMETER (Slave device) */
  *pBuffer = SPIx_WriteRead(DUMMY_BYTE);
  NumByteToRead--;
  pBuffer++;
 }

 /* Set chip select High at the end of the transmission */
 ACCELERO_CS_HIGH();
}
```

set - multibyte bit if >1 byte

- indicate read (device to supply data)

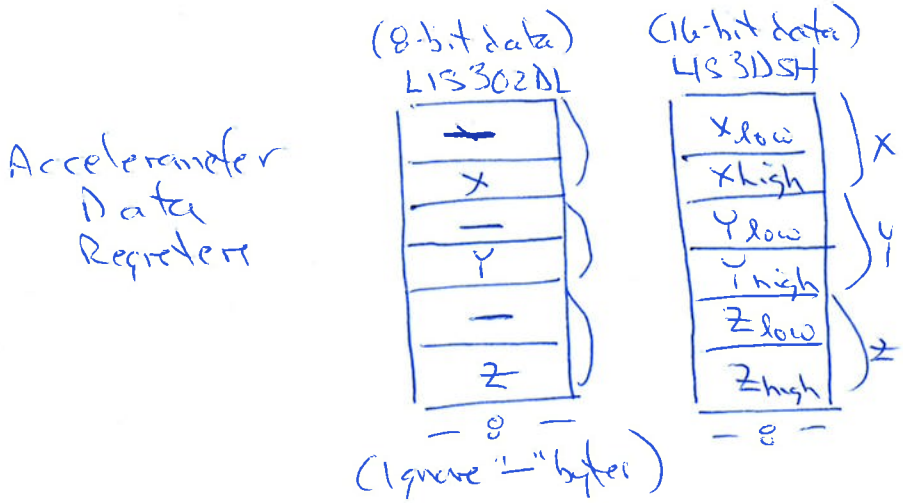
) CS low (active)

send read addr. + "dummy" byte

return byte from device

return value

) CS high (inactive)



Accelerometer Data Register

```
/**
  * @brief  Writes one byte to the Accelerometer.
  * @param  pBuffer: pointer to the buffer containing the data to be written to the Accelerometer.
  * @param  WriteAddr: Accelerometer's internal address to write to.
  * @param  NumByteToWrite: Number of bytes to write.
  * @retval None
  */
void ACCELERO_IO_Write(uint8_t *pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite)
{
  /* Configure the MS bit:
     - When 0, the address will remain unchanged in multiple read/write commands.
     - When 1, the address will be auto incremented in multiple read/write commands.
  */
  if(NumByteToWrite > 0x01)
  {
    WriteAddr |= (uint8_t)MULTIPLEBYTE_CMD;
  }
  /* Set chip select Low at the start of the transmission */
  ACCELERO_CS_LOW();

  /* Send the Address of the indexed register */
  SPIx_WriteRead(WriteAddr);

  /* Send the data that will be written into the device (MSB First) */
  while(NumByteToWrite >= 0x01)
  {
    SPIx_WriteRead(*pBuffer);
    NumByteToWrite--;
    pBuffer++;
  }

  /* Set chip select High at the end of the transmission */
  ACCELERO_CS_HIGH();
}
```

- set "multi-byte" bit, if needed

CS low for SPI mode

- send write addr.
- send byte(s)
- ignore returned data

CS back high

```
235  /**
236    * @brief  Get XYZ axes acceleration.
237    * @param  pDataXYZ: Pointer to 3 angular acceleration axes.
238    *                   pDataXYZ[0] = X axis, pDataXYZ[1] = Y axis, pDataXYZ[2] = Z axis
239    * @retval None
240    */
241  void BSP_ACCELERO_GetXYZ(int16_t *pDataXYZ)
242  {
243    int16_t SwitchXY = 0;
244
245    if(AcceleroDrv->GetXYZ != NULL)
246    {
247      AcceleroDrv->GetXYZ(pDataXYZ);
248
249      /* Switch X and Y Axes in case of LIS302DL MEMS */
250      if(AcceleroDrv == &Lis302dlDrv)
251      {
252        SwitchXY  = pDataXYZ[0];
253        pDataXYZ[0] = pDataXYZ[1];
254        /* Invert Y Axis to be conpliant with LIS3DSH MEMS */
255        pDataXYZ[1] = -SwitchXY;
256      }
257    }
258  }
```

-calls LIS302DL_ReadACC or LIS3DSH_ReadACC

```
/* LIS302DL struct */


/* Initalization struct */
typedef struct
{
 uint8_t Power_Mode;            /* Power-down/Active Mode */
 uint8_t Output_DataRate;       /* OUT data rate 100 Hz / 400 Hz */
 uint8_t Axes_Enable;           /* Axes enable */
 uint8_t Full_Scale;            /* Full scale */
 uint8_t Self_Test;             /* Self test */
}LIS302DL_InitTypeDef;


/* Interrupting configuration struct */
typedef struct
{
 uint8_t Latch_Request;          /* Latch interrupt request into CLICK_SRC register*/
 uint8_t SingleClick_Axes;       /* Single Click Axes Interrupts */
 uint8_t DoubleClick_Axes;       /* Double Click Axes Interrupts */
}LIS302DL_InterruptConfigTypeDef;


/* High Pass Filter struct */
typedef struct
{
 uint8_t HighPassFilter_Data_Selection;    /* Internal filter bypassed or data from internal filter send to
output register*/
 uint8_t HighPassFilter_CutOff_Frequency;  /* High pass filter cut-off frequency */
 uint8_t HighPassFilter_Interrupt;         /* High pass filter enabled for Freefall/WakeUp #1 or #2 */
}LIS302DL_FilterConfigTypeDef;
```