1 Introduction

In this document, you will learn about the LIS3DH 3-axis accelerometer chip, the principles behind how MEMS accelerometers work, as well as the specifics of how to configure and use the LIS3DH with an ATMega328p.

Contents

2 What is MEMS?

MEMS (Micro-electro-mechanical systems) is a blanket term for the technologies behind microscopic devices, which often include moving parts. These devices can be very small, often with features on the order of tens of micrometers in size.

Creating MEMS devices is an intricate process, involving techniques from the world of semiconductor device fabrication. The moving parts and circuits are carved away from a thin slab of material by electrical arcs, as in Electrical Discharge Machining (EDM), or else are eaten away with chemical reactions, as in wet or dry-etching techniques. For complex MEMS devices, multiple passes are required in order to remove the material layer-by-layer until only the desired structures are left.

2.1 MEMS Accelerometers

MEMS sensors are often built as capacitative switches, which will change their capacitance in response to stresses on a moving element. For an accelerometer, free-floating arms are attached to a capactative half-bridge, and when the arm masses move, the half-bridge is displaced or bent slightly, causing an imbalance in its capacitance. Because the arm masses are known with high precision, the change in acceleration can be measured accurately by observing the change in capacitance over the half-bridge.

2.2 The LIS3DH's sensing element

The LIS3DH has a proprietary design for its sensing element, but thr design is based on the same principles as other MEMS accelerometers. To quote directly from the data sheet:

> When an acceleration is applied to the sensor the proof mass displaces from its nominal position, causing an imbalance in the capacitive half-bridge. This imbalance is measured using charge integration in response to a voltage pulse applied to the capacitor. At steady state the nominal value of the capacitors are few pF and when an acceleration is applied the maximum variation of the capacitive load is in the fF range.

3 SPI Interface

The LIS3DH has two interfacing options: SPI, and I^2C. In this guide, we'll be covering just the SPI interfacing.

3.1 SPI clock settings

3.2 Ensuring SPI Reads work

To ensure you've implemented SPI reads correctly, you'll want to attempt to read the `WHO_AM_I` register (address `0x0F`), which serves to uniquely identify the LIS3DH on a multi-sensor bus, and will always return the value `0x33` when read.

If your accelerometer read returns a value other than `0x33`, you know your implementation or wiring is wrong!

3.3 Configuring the LIS3DH

The LIS3DH's register map is comprised almost entirely of configuration and status registers, with a few registers reserved for outputs. As a result of this (and a lack of guidance in the datasheet and related materials), it can be intimidating to try to configure the LIS3DH from scratch.

Based on trial and error, and much time spent poring over the data sheet, this is the recommended startup sequence for collecting 3-axis acceleration data, with "data ready" interrupts:

1. Write `(ODR | 0x07)` → `CTRL_REG1 (0x20)` :: Configure output data rate (ODR) + X, Y, Z-axis enable bits.
2. Write `0x08` → `CTRL_REG5 (0x24)` :: Disable FIFO + enable latching interrupts.
3. Write `0x00` → `FIFO_CTRL_REG (0x24)` :: Set FIFO system to Bypass mode.
4. Write `0x10` → `CTRL_REG3 (0x22)` :: Enable DRDY1 interrupt on the LIS3DH's INT1 pin.
5. Write `0x02` → `CTRL_REG6 (0x25)` :: Set the INT1 interrupt to be active low.

Warning: The accelerometer will retain its configuration as long as it has power. This means that if you hit the reset switch on the ATMega, the LIS3DH will not be reset! To ensure the LIS3DH comes up in the correct configuration each time the ATMega is reset, you should explicitly set these configuration values as part of your initialization code, before the main loop.

The other configuration registers, such as those controlling event-triggered interrupts and the FIFO subsystem, are available to be explored by the adventurous, but will not be covered here.

4 LIS3DH interfacing from the ATMega328P

4.1 SPI Configuration

`SPCR` Field	Recommended Value	Remark
`SPIE`	0	Disable SPI Interrupts
`SPE`	1	Enable SPI
`DORD`	0	MSB-first
`MSTR`	1	Master mode
`CPOL`	0	Leading edge rising, trailing edge falling
`CPHA`	0	Leading edge sample, trailing edge setup
`SPR[1:0]`	11_2	$f_{\text{sck}} = \frac{f_{\text{osc}}}{128}$

Figure 1: Table showing suggested SPI configuration for interfacing the LIS3DH with an ATMega328P. Keep in mind that a separate GPIO pin must also be used as the SS signal.

4.2 LIS3DH Registers

Like most SPI devices, the LIS3DH can be thought of as a linear array of registers, some of which can be written, some which can be read, and some which are both readable and writable. Be careful to keep in mind that the

LIS3DH's registers can only be accessed over SPI, as distinct from the ATMega328P's registers, which can be accessed as memory locations on the AVR.

For example, `CTRL_REG1 = 0x20;` is **invalid**, you instead must write the address `CTRL_REG1` via SPI, then the data value you wish to store in it.

4.3 How to Read a Register

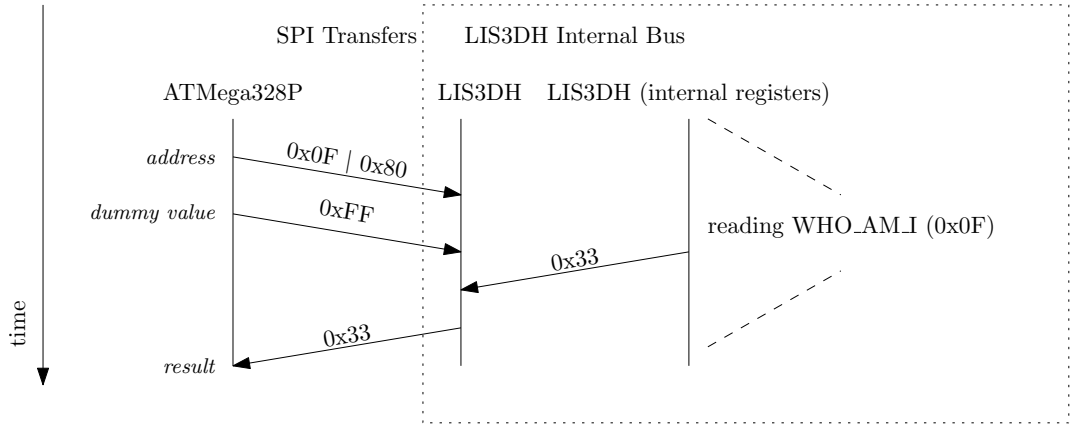

Figure 2: Diagram showing a visual representation of reading the WHO_AM_I register of the LIS3DH over SPI.

Algorithm 1: Pseudocode to read the data from a LIS3DH register.

Pull $\overline{SS}$ low
// set address read flag in the 8 bit byte, remember a register address is 6 bits internally
Write register_address | 0x80 via SPI
Wait for transfer to complete
Write a dummy value (e.g. 0xFF) via SPI
Wait for the transfer to complete
Save the contents of the SPI data register somewhere – this is the result sent from the LIS3DH
Pull $\overline{SS}$ high

4.4 How to Write a Register

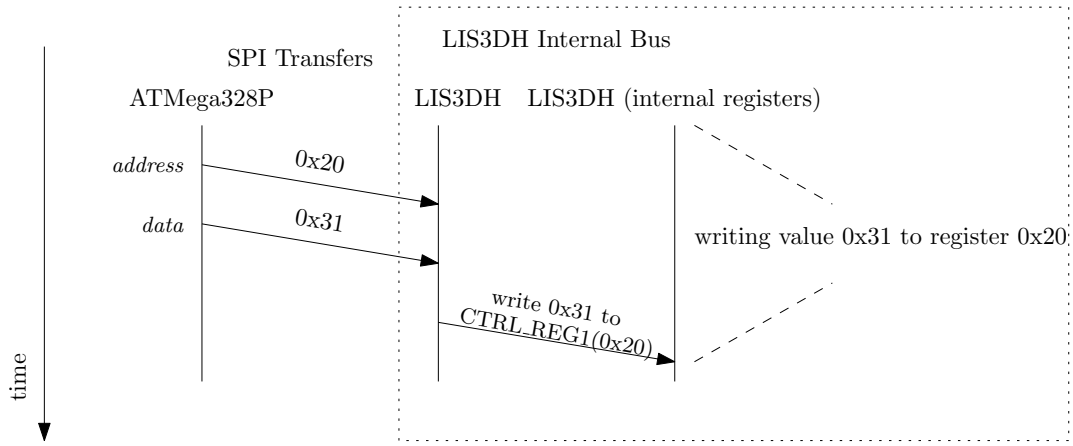

Figure 3: Diagram showing a visual representation of writing the value 0x31 to register 0x20 on the LIS3DH over SPI.

Algorithm 2: Pseudocode showing how to write to a LIS3DH register.

Pull $\overline{SS}$ low
Write register_address via SPI *// note that we don't OR with 0x80 when writing*
Wait for transfer to complete
Write desired value via SPI
Wait for the transfer to complete
Pull $\overline{SS}$ high

4.5 Interrupts

You should use `INT0` to trigger an ISR to run whenever the LIS3DH has data ready (hint: `ISR (INT0_vect) { ... }`). Remember to read from the `INT1_SOURCE` register on the LIS3DH to clear the interrupt on the device.

References

- ATMega328P data sheethttps://www.microchip.com/wwwproducts/en/ATmega328p
- LIS3DH data sheethttps://www.st.com/resource/en/datasheet/lis3dh.pdf