

# C-CODE EXAMPLES FOR CMA3000 ACCELEROMETER

## 1 INTRODUCTION

This technical note presents simple C-code examples for SPI communication between the CMA3000 sensor and a MCU.

Please refer to the document "CMA3000-D0X Product Family Specification 8281000" for further information on CMA3000 register addressing and SPI communication. The examples are tested with CMA3000 DEMO KIT hardware. Please refer to the document "CMA3000 DEMO KIT User Manual 8288400" for further information about the CMA3000 DEMO KIT. This document applies to CMA3000-D01.

## 2 C-CODE EXAMPLE

This document covers the following C-code examples listed below:

- 2.1 Read CMA3000 register
- 2.2 Write CMA3000 register

## 2.1 Read CMA3000 register

```

/* This example reads a specified register of the CMA3000.
 * The register address is given as a parameter.
 * NOTE: The command byte of SPI bus is :
 *      (MSB) A A A A A A A RW 0 (LSB)
 *
 *      |           | | *- always zero
 *      |           | *--- Read / Write Bit
 *      |-----*----- Register address
 *
 * so the real byte sent to SPI bus must be calculated:
 *      BYTE = (Address * 4) + RW*2;
 * For Read operation RW = 0 and for Write operation RW = 1
 * 'Address *4' shifts the bit pattern to left by 2 and 'RW*2' by 1.
 * For example register = 0x06
 *      BYTE = 0x06*4 + 0*2;
 *      BYTE = 0x18;
 * This calculation is done inside the function.
 *
 * Usage of the following function:
 *
 * int Xacc;
 *
 * Xacc = CMA3000_read(0x06);
 *
 * This example has been written for an Atmel ATmega168 processor in a GCC environment.
 * Other processors or environments might need different port names or SPI device names.
 * SPDR - is the read/write data to/from SPI bus register
 * (see ATmega168 document (doc2545.pdf) pages 159 - 167 for more information) */

#define SPI_CSB 0x04 // This is PB2 at the port
#define SPI_PORT PORTB

// Function takes in the 8-bit register address and returns 8-bit register value.
int CMA3000_Read(unsigned int Address)
{
    int result;
    result = 0;
    Address = Address << 2; // RW bit is set to zero by shifting the bit
                          // pattern to left by 2

    SPI_PORT = SPI_PORT & (~SPI_CSB); // Set CSB to zero
    SPDR = Address; // Write command to SPI bus

    while(!(SPSR & (1 << SPIF))); // Wait until data has been sent

    SPDR = 0x00; // Send dummy data to enable next 8 bit clocking
    while(!(SPSR & (1 << SPIF))); // Wait until data has been sent

    result = SPDR; // Get the result

    SPI_PORT = SPI_PORT | SPI_CSB; // Set CSB to one
    return result;
}

```

## 2.2 Write CMA3000 register

```

/* This example writes a specified register of the CMA3000.
 * The register address and data to be written are given as parameters.
 * NOTE: The command byte of SPI bus is :
 *      (MSB) A A A A A A RW 0 (LSB)
 *      |           | | * - always zero
 *      |           | * - - - Read / Write Bit
 *      |-----*----- Register address
 *
 * so the real byte sent to SPI bus must be calculated:
 *      BYTE = (Address * 4) + RW*2;
 * For Read operation RW = 0 and for Write operation RW = 1
 * 'Address *4' shifts the bit pattern to left by 2 and 'RW*2' by 1.
 * For example register = 0x02
 *      BYTE = 0x02*4 + 1*2;
 *      BYTE = 0x0A;
 * This calculation is done inside the function.
 *
 * Usage of the following function:
 *
 * CMA3000_write(0x02,0x00);
 *
 * This example has been written for an Atmel ATmega168 processor in a GCC environment.
 * Other processors or environments might need different port names or SPI device names.
 * SPDR - is the read/write data to/from SPI bus register
 * (see ATmega168 document (doc2545.pdf) pages 159 - 167 for more information) */

#define SPI_CSB 0x04 // This is PB2 at the port
#define SPI_PORT PORTB

// Function takes in the 8-bit register address and 8-bit register value.
void CMA3000_Write(unsigned int Address, unsigned char Data)
{
    Address = Address << 2; // RW bit is set to zero by shifting the bit
                          // pattern to left by 2
    Address |= 0x02; // Set write bit

    SPI_PORT = SPI_PORT & (~SPI_CSB); // Set CSB to zero
    SPDR = Address; // Write command to SPI bus

    while(!(SPSR & (1 << SPIF))); // Wait until data has been sent

    SPDR = Data; // Write data to SPI bus

    while(!(SPSR & (1 << SPIF))); // Wait until data has been sent

    SPI_PORT = SPI_PORT | SPI_CSB; // Set CSB to one
    return;
}

```