# Application Note

## MLX90427
## User Manual on SPI Commands

## Contents

# Application Note

## MLX90427
## User Manual on SPI Commands

## 1. Scope

This document acts as a manual for the MLX90427, showing the implementation of the command set and how to use them. The MOSI commands can be distinguished by an opcode, of which the list is shown in section 3. Sections 4 and 6 show the format of the MOSI and MISO messages.

## 2. SPI

### 2.1. SPI Mode

The SPI mode for the MLX90427 is “Mode 0”, so CPOL=0 and CPHA=0. Data is captured on the rising edge of the clock and updated on the falling edge of the clock.

The SPI protocol is implementing a **full-duplex** communication. **The MISO message during a MOSI message is the answer to the previous MISO message**. For examples, please check section 7.

### 2.2. SPI Format

All commands have a length of 8 bytes. The bytes are sent MSB first (Byte 7, bit 7), and are shown throughout the document in the formats below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte 7 | | | | | | | | Byte 6 | | | | | | | |
| Byte 5 | | | | | | | | Byte 4 | | | | | | | |
| Byte 3 | | | | | | | | Byte 2 | | | | | | | |
| Byte 1 | | | | | | | | Byte 0 (CRC[7:0]) | | | | | | | |

Byte 0 is always a CRC-8 calculated on the full message (section 4.1). Byte 1 contains in most cases (except RESULT_DATA and RESULT_MEAS) an opcode of 7 bits, and in that case the MSbit of Byte 1 is 0.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSB | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | LSB |

Sections in blue show parameters for that function.

## 3. Overview of Commands (opcodes + timings)

The below table provides an overview of the commands and the linked opcode, to be used in Byte 1 of the SPI commands. Note that these are only valid for the MOSI messages, MISO messages can repeat the opcode from the MOSI message. Also the expected response is shown, for reference. The processing time shown is the time from SPI end-of-frame to result (RESULT_ACK, RESULT_STATUS) ready. In case of sending the following message faster than 40µs after the previous SPI end-of-frame, the output can be showing garbage data. For the measurements, the processing time is different depending on the mode. This is elaborated in section 5.3.3.

| MOSI Command | Opcode | Expected MISO Response | Processing time (max, µs) |
|---|---|---|---|
| GET | 0x07 | RESULT_DATA | 90 |
| GET_NEXT | 0x0B | RESULT_DATA | 90 |
| SET | 0x0D | RESULT_ACK | 120 |
| NOP | 0x13 | RESULT_STATUS | 100 |
| RST | 0x15 | - | 80 |
| STBY | 0x16 | - | 100 |
| TRG_NORMAL | 0x19 | RESULT_MEAS | cfr. section 5.3.3 |
| TRG_SYNC | 0x1A | RESULT_MEAS | cfr. section 5.3.3 |
| PROTECTED_MODE | 0x23 | RESULT_ACK | 100 |
| EXIT | 0x25 | RESULT_ACK | 90 |
| NVM_RECALL | 0x26 | RESULT_ACK | 80 |
| NVM_STORE | 0x29 | RESULT_ACK | 13200 |
| READ | 0x2A | RESULT_DATA | 110 |
| READ_NEXT | 0x2C | RESULT_DATA | 100 |
| WRITE | 0x31 | RESULT_ACK | 110 |
| WRITE_NEXT | 0x32 | RESULT_ACK | 100 |
| RESET_PARTIAL | 0x34 | - | 80 |

Wait at least 3ms after a TRG_NORMAL or TRG_SYNC+sync pulse before executing a NVM_STORE.

## 4. CRC

Two types of CRC are being used in the IC. One for the SPI communication, a CRC-8. Another to protect the NVRAM, a CRC-16.

### 4.1. SPI CRC-8

The communication is protected with a CRC-8, using polynomial 0x2F ($x^8+x^5+x^3+x^2+x+1$), initialization value 0xFF, and final XOR value 0x00.

A routine to calculate this CRC is found below:

```
uint8_t CRC_2F(uint8_t* message)
{
    uint8_t crc = 0xFF; // initialization value
    uint8_t i, j;
    for (i = 0; i < 7; i++) {
        crc = crc ^ message[i];
        for (j = 0; j < 8; j++) {
            if ((crc & 0x80) == 0)
                crc = crc << 1;
            else
                crc = (uint8_t)((crc << 1) ^ 0x2F);
        }
    }
    crc = crc ^ 0x00; // final XOR value
    return crc;
}
```

The value can also be calculated via a table and routine below:

```
2F_TABLE[256] = {
    0x00, 0x2F, 0x5E, 0x71, 0xBC, 0x93, 0xE2, 0xCD,
    0x57, 0x78, 0x09, 0x26, 0xEB, 0xC4, 0xB5, 0x9A,
    0xAE, 0x81, 0xF0, 0xDF, 0x12, 0x3D, 0x4C, 0x63,
    0xF9, 0xD6, 0xA7, 0x88, 0x45, 0x6A, 0x1B, 0x34,
    0x73, 0x5C, 0x2D, 0x02, 0xCF, 0xE0, 0x91, 0xBE,
    0x24, 0x0B, 0x7A, 0x55, 0x98, 0xB7, 0xC6, 0xE9,
    0xDD, 0xF2, 0x83, 0xAC, 0x61, 0x4E, 0x3F, 0x10,
    0x8A, 0xA5, 0xD4, 0xFB, 0x36, 0x19, 0x68, 0x47,
    0xE6, 0xC9, 0xB8, 0x97, 0x5A, 0x75, 0x04, 0x2B,
    0xB1, 0x9E, 0xEF, 0xC0, 0x0D, 0x22, 0x53, 0x7C,
    0x48, 0x67, 0x16, 0x39, 0xF4, 0xDB, 0xAA, 0x85,
    0x1F, 0x30, 0x41, 0x6E, 0xA3, 0x8C, 0xFD, 0xD2,
    0x95, 0xBA, 0xCB, 0xE4, 0x29, 0x06, 0x77, 0x58,
    0xC2, 0xED, 0x9C, 0xB3, 0x7E, 0x51, 0x20, 0x0F,
    0x3B, 0x14, 0x65, 0x4A, 0x87, 0xA8, 0xD9, 0xF6,
    0x6C, 0x43, 0x32, 0x1D, 0xD0, 0xFF, 0x8E, 0xA1,
    0xE3, 0xCC, 0xBD, 0x92, 0x5F, 0x70, 0x01, 0x2E,
    0xB4, 0x9B, 0xEA, 0xC5, 0x08, 0x27, 0x56, 0x79,
    0x4D, 0x62, 0x13, 0x3C, 0xF1, 0xDE, 0xAF, 0x80,
    0x1A, 0x35, 0x44, 0x6B, 0xA6, 0x89, 0xF8, 0xD7,
    0x90, 0xBF, 0xCE, 0xE1, 0x2C, 0x03, 0x72, 0x5D,
    0xC7, 0xE8, 0x99, 0xB6, 0x7B, 0x54, 0x25, 0x0A,
    0x3E, 0x11, 0x60, 0x4F, 0x82, 0xAD, 0xDC, 0xF3,
    0x69, 0x46, 0x37, 0x18, 0xD5, 0xFA, 0x8B, 0xA4,
    0x05, 0x2A, 0x5B, 0x74, 0xB9, 0x96, 0xE7, 0xC8,
    0x52, 0x7D, 0x0C, 0x23, 0xEE, 0xC1, 0xB0, 0x9F,
    0xAB, 0x84, 0xF5, 0xDA, 0x17, 0x38, 0x49, 0x66,
    0xFC, 0xD3, 0xA2, 0x8D, 0x40, 0x6F, 0x1E, 0x31,
    0x76, 0x59, 0x28, 0x07, 0xCA, 0xE5, 0x94, 0xBB,
    0x21, 0x0E, 0x7F, 0x50, 0x9D, 0xB2, 0xC3, 0xEC,
    0xD8, 0xF7, 0x86, 0xA9, 0x64, 0x4B, 0x3A, 0x15,
    0x8F, 0xA0, 0xD1, 0xFE, 0x33, 0x1C, 0x6D, 0x42
};
```

```
crc = 0xFF; // initialization value
crc = 2F_TABLE[Byte7 ^ crc];
crc = 2F_TABLE[Byte6 ^ crc];
crc = 2F_TABLE[Byte5 ^ crc];
crc = 2F_TABLE[Byte4 ^ crc];
crc = 2F_TABLE[Byte3 ^ crc];
crc = 2F_TABLE[Byte2 ^ crc];
crc = 2F_TABLE[Byte1 ^ crc];
crc = crc ^ 0x00; // final XOR value
```

## 4.2. NVM CRC-16

The NVRAM is protected with a CRC-16, using polynomial 0x1021 ($x^{16}+x^{12}+x^5+1$), initialization value 0xFFFF, and final XOR value 0x0000. The CRC corresponds to the AUTOSAR 16-bit CCITT-FALSE CRC16 as defined in "*AUTOSAR, "Specification of CRC Routines", R20-11, Nov. 2020"*.

The customer section of the NVRAM is ranging from address 0x1000 to 0x1058, with 0x1058 containing the CRC-16 calculated on the range 0x1000 to 0x1056, so a length of 44 words. A calculation routine is similar to the one for the CRC-8, the words are split into bytes and the calculation starts on the MSByte.

```
uint16_t CRC_1021(uint16_t* nvram)
{
    uint16_t crc = 0xFFFF;
    uint8_t i, j;
    for (i = 0; i < 44; i++) {
        crc = crc ^ (uint16_t)(nvram[i] & 0xFF00);
        for (j = 0; j < 8; j++) {
            if ((crc & 0x8000) == 0) {
                crc = crc << 1;
            } else {
                crc = (uint16_t)((crc << 1) ^ 0x1021);
            }
        }
        crc = crc ^ (uint16_t)((nvram[i] & 0x00FF) << 8);
        for (j = 0; j < 8; j++) {
            if ((crc & 0x8000) == 0) {
                crc = crc << 1;
            } else {
                crc = (uint16_t)((crc << 1) ^ 0x1021);
            }
        }
    }
    return crc;
}
```

# 5. MOSI Messages

## 5.1. Control Commands

### 5.1.1. NOP

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td>0</td><td colspan="7">0x13 (NOP)</td><td colspan="8">0x4A</td></tr>
</table>

A “no operation” command, typically used to get system’s current status information.

Expected response: RESULT_STATUS

### 5.1.2. RST

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x1F</td><td colspan="8">0x4C</td></tr>
<tr><td>0</td><td colspan="7">0x15 (RST)</td><td colspan="8">0xD6</td></tr>
</table>

Reset command, to warm reset the system. System will go through a safe startup phase after the warm reset. **0x1F4C** acts as a key.

Expected response: none

### 5.1.3. STBY

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x6B</td><td colspan="8">0x8C</td></tr>
<tr><td>0</td><td colspan="7">0x16 (STBY)</td><td colspan="8">0xF2</td></tr>
</table>

Requests the system to go in standby mode. **0x6B8C** acts as a key.

Expected response: none

[1] *These 0x00 values could be replaced by any other value, but make sure to update the CRC accordingly.*

### 5.1.4. PROTECTED_MODE

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0xB2 | | | | | | | | 0x55 | | | | | | | |
| 0xA2 | | | | | | | | 0xD3 | | | | | | | |
| 0x8C | | | | | | | | 0x5E | | | | | | | |
| 0 | 0x23 (PROTECTED_MODE) | | | | | | | 0x0D | | | | | | | |

Opens a protected mode session. Activating such a session allows access to the commands to read and write the NVRAM of the IC. **0xB255A2D38C5E** acts as a key.

Expected response: RESULT_ACK

### 5.1.5. EXIT

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x25 (EXIT) | | | | | | | 0x75 | | | | | | | |

Exits the protected mode session, without performing a system reset. Note that this command only makes sense when in a protected mode session.

Expected response: RESULT_ACK

### 5.1.6. RST_PARTIAL

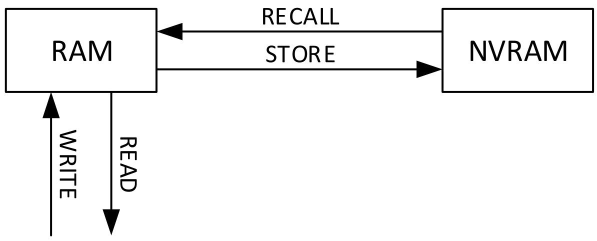
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x6C | | | | | | | | 0xF0 | | | | | | | |
| 0 | 0x34 (RST_PARTIAL) | | | | | | | 0x5E | | | | | | | |

Partial reset command. System will be reset, but the NVRAM recall is bypassed in the startup sequence. **0x6CF0** acts as a key.

Expected response: none

## 5.2. Memory Commands

The memory of the MLX90427 is an NVRAM. This architecture allows that all read and write operations happen in a volatile memory (after a reset of the system, the content is reverted). To store content permanently, a store operation is required, which copies all the content from the volatile memory in the non-volatile memory (the NVRAM). A recall operation copies the contents in the other direction, and is also performed on every system reset (any type). **All commands of this section require the protected mode to be active. Only the customer area of the NVRAM can be written to, the MLX area is protected against writing operations.**




### 5.2.1. READ and READ_NEXT

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | LENGTH[7:0] | | | | | | | |
| ADDRESS[15:0] | | | | | | | | | | | | | | | |
| 0 | 0x2A (READ) | | | | | | | CRC[7:0] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x2C (READ_NEXT) | | | | | | | 0x0D | | | | | | | |

Read from system memory. **LENGTH** denotes the number of words (16-bit) to read, starting at the specified **ADDRESS**. Returned data is of the type "RESULT_DATA" and can contain up to 3 words. In case LENGTH is larger than 3, this command should be followed by READ_NEXT to read out the remaining requested words. In case LENGTH is not a multiple of 3, remaining words are 0x0000.

Expected response: RESULT_DATA.

### 5.2.2. WRITE and WRITE_NEXT

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">DATA[15:0]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">LENGTH[7:0]</td></tr>
<tr><td colspan="16">ADDRESS[15:0]</td></tr>
<tr><td>0</td><td colspan="7">0x31 (WRITE)</td><td colspan="8">CRC[7:0]</td></tr>
</table>

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="16">DATA0[15:0]</td></tr>
<tr><td colspan="16">DATA1[15:0]</td></tr>
<tr><td colspan="16">DATA2[15:0]</td></tr>
<tr><td>0</td><td colspan="7">0x32 (WRITE_NEXT)</td><td colspan="8">CRC[7:0]</td></tr>
</table>

Write **DATA**/**DATA0**/**DATA1**/**DATA2** to system memory. **LENGTH** denotes the number of words (16-bit) to write, starting at the specified **ADDRESS**. Similar to the READ command, in case LENGTH is larger than 1, this command should be followed by WRITE_NEXT to write the remaining words. If LENGTH-1 is not a multiple of 3, remaining words are not taken into account.

Expected response: RESULT_ACK.

### 5.2.3. NVM_RECALL

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td colspan="8">0x00[1]</td><td colspan="8">0x00[1]</td></tr>
<tr><td>0</td><td colspan="7">0x26 (NVM_RECALL)</td><td colspan="8">0x04</td></tr>
</table>

Trigger an NVRAM recall operation.

Expected response: RESULT_ACK.

### 5.2.4. NVM_STORE (with optional lock)

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">0xC8</td><td colspan="8">0xF4</td></tr>
<tr><td colspan="8">0x77</td><td colspan="8">0x84</td></tr>
<tr><td colspan="8">0xCE/0x43</td><td colspan="8">0x83/0xE6</td></tr>
<tr><td>0</td><td colspan="7">0x29 (NVM_STORE)</td><td colspan="8">0xE9/0x80</td></tr>
</table>

Trigger an NVRAM store operation. **The operation will fail in case the CRC-16 (section 4.2) of the NVRAM is not correctly recalculated and written prior to this operation**. Using the values in *italic* will enable a memory lock, where **0xC8F47784CE83/*43E6*** acts as a key. After locking, changing the parameters in the NVRAM is not possible anymore. To enable the lock, calculate the CRC-16 of the NVRAM taking the lock flag into account, else the procedure will exit with an error. Note that this command will take significant time to execute (13.2ms), and if communication starts too soon, ERR_ONGOING will be returned.

Expected response: RESULT_ACK

## 5.3. Measurement Commands

There are two ways of making measurements with the MLX90427: a measurement triggered by a command or one triggered by a sync pulse. The last option allows for multiple devices to acquire a measurement at the same time.

### 5.3.1. TRG_NORMAL

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">0x00(1)</td><td colspan="8">0x00(1)</td></tr>
<tr><td colspan="8">0x00(1)</td><td colspan="8">TRIG-to-READ TIMEOUT[7:0]</td></tr>
<tr><td colspan="4">MODE[3:0]</td><td colspan="4">SEL[3:0]</td><td>RC</td><td colspan="7">0x00(1)</td></tr>
<tr><td>0</td><td colspan="7">0x19 (TRG_NORMAL)</td><td colspan="8">CRC[7:0]</td></tr>
</table>

Triggers and configures a normal acquisition and processing cycle. The answer is of type "RESULT_MEAS". In case **RC** is set, the measurement counter is reset to 1 again. With **MODE**, the measurement mode is configured. If **SEL**>0, it determines the secondary data, that can be returned in the RESULT_MEAS message. Otherwise the secondary data is the default mentioned in the table below. The MSBits are padded with 0 in case the secondary data is less than 16 bits in length.

MODE and SEL must be consistent. If this is not the case ERR_ARGS will be returned.

A **TIMEOUT** is available, encoding the max allowed time between this command and the next command (reading the results). Setting it to 0x0 disables the mechanism, else the timeout equals:

$$timeout(\mu s) = 1100 + 100 \cdot code$$

| MODE | Description | DATA_PRIMARY | DATA_SECONDARY (SEL=0x0) |
|---|---|---|---|
| 0x1 | Legacy rotary | ANGLE[13:0] | 0x0000 |
| 0x2 | Stray field immune rotary (dBz) | ANGLE_DBZ[13:0] | 0x0000 |
| 0x3 | Dual mode rotary | ANGLE[13:0] | ANGLE_DBZ[13:0] |
| 0x4 | Diagnostic rotary (full HP + half HP) | ANGLE[13:0] | ANGLE_HALF[13:0] |
| 0x5 | Fields 2D legacy | FIELD_B0[13:0] | FIELD_B1[13:0] |
| 0x6 | Fields 2D dBz | FIELD_B0_DBZ[13:0] | FIELD_B1_DBZ[13:0] |
| 0x7 | Full diagnostic sequence (FDS)[2] | N/A | N/A |
| 0x9 | Joystick **(only in joystick config)** | ALPHA[13:0] | PUSH_BUTTON[1:0] \| BETA[13:0] |
| 0xE | Fields 3D[3] **(only in joystick config)** | N/A | N/A |

*[2] For "FDS" mode, RESULT_STATUS instead of RESULT_MEAS returned. GET commands are needed to access the data.*
*[3] For "Fields 3D" mode, RESULT_MEAS_3D returned, see specific SPI format description (Section 6.1.2).*

| SEL | Description | DATA_SECONDARY |
|---|---|---|
| 0x0 | Default | According to MODE |
| 0x1 | Field B0, legacy | FIELD_B0[13:0] |
| 0x2 | Field B1, legacy | FIELD_B1[13:0] |
| 0x3 | Field B0, dBz | FIELD_B0_DBZ[13:0] |
| 0x4 | Field B1, dBz | FIELD_B1_DBZ[13:0] |
| 0x6 | Field magnitude estimation, legacy | FIELD_MAG_LEG[15:0] |
| 0x7 | Field magnitude estimation, dBz | FIELD_MAG_DBZ[15:0] |
| 0x8 | Virtual gain (AGC) | GAIN_DBZ[7:0] \| GAIN[7:0] |
| 0x9 | Rough offset correction, legacy | ROC_SIN[7:0] \| ROC_COS[7:0] |
| 0xA | Rough offset correction, dBz | ROC_SIN_DBZ[7:0] \| ROC_COS_DBZ[7:0] |

Expected response: RESULT_MEAS(2)(3).

### 5.3.2. TRG_SYNC

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">0x00(1)</td><td colspan="8">SYNC-to-SYNC TIMEOUT[7:0]</td></tr>
<tr><td colspan="8">0x00(1)</td><td colspan="8">SYNC-to-READ TIMEOUT[7:0]</td></tr>
<tr><td colspan="4">MODE[3:0]</td><td colspan="4">SEL[3:0]</td><td>RC</td><td colspan="7">0x00(1)</td></tr>
<tr><td>0</td><td colspan="7">0x1A (TRG_SYNC)</td><td colspan="8">CRC[7:0]</td></tr>
</table>

The command structure (**MODE**, **SEL**, **RC**) and content is similar to TRG_NORMAL (5.3.1).

Sending this command arms the IC to wait for a sync pulse in order to start the acquisition. The sync pulse is a negative pulse on the CS line, (min 20µs, max 400µs) in length. A second sync pulse will be considered an invalid SPI message.

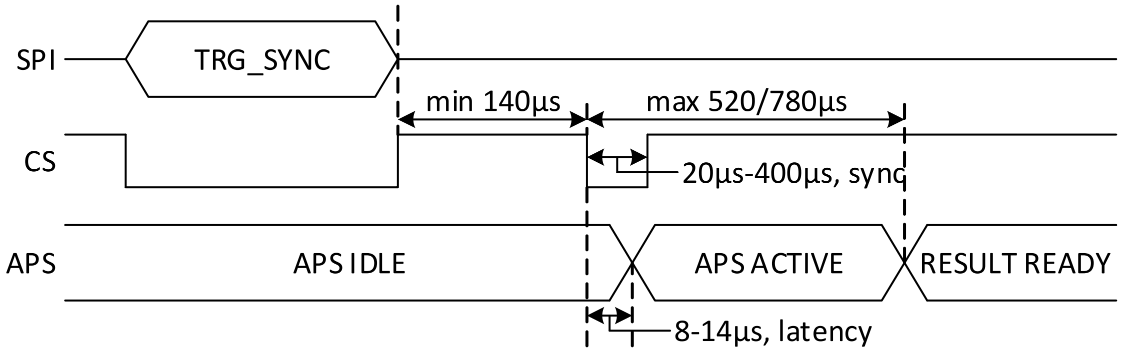
The **TIMEOUT**S are two-fold, one for timeouts between sync pulse and readback of the data, another one for the time between sync pulses. Setting them to 0x0 disables the respective timeout, else the timeout equals:

$$timeout(\mu s) = 1100 + 100 \cdot code$$

Expected response: RESULT_MEAS(2)(3).

### 5.3.3. Timing for Measurement Commands

| Mode | SPI end-of-frame to APS start (latency, µs) | | SPI end-of-frame to result ready (µs) |
|---|---|---|---|
| NORMAL legacy | 140 | | 610 |
| NORMAL dBz | 140 | | 610 |
| NORMAL dual mode | 140 | | 900 |
| NORMAL diagnostic | 140 | | 900 |
| NORMAL FDS | 140 | | 940 |
| NORMAL joystick | 140 | | 860 |
| **Mode** | **SPI end-of-frame to waiting for SYNC (µs)** | **SYNC to APS start (latency, µs)** | **SYNC to result ready (µs)** |
| SYNC legacy | 140 | 8 to 14 | 520 |
| SYNC dBz | 140 | 8 to 14 | 520 |
| SYNC dual mode | 140 | 8 to 14 | 780 |
| SYNC diagnostic | 140 | 8 to 14 | 780 |



### 5.4. Get Commands

These commands are used to get specific information, data, functions, or parameters from the IC. The commands GET and GET_NEXT look like below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| GET_SEL[7:0] | | | | | | | | 0x00[1] | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| 0 | 0x0B (GET_NEXT) | | | | | | | 0xB3 | | | | | | | |

Sends a GET command to the IC. Depending on **GET_SEL**, a different content is returned via RESULT_DATA MISO messages. Due to the fact that some data is longer than 3 words, GET_NEXT is required for these. Unused words are set to 0x0000. DATA containing values shorter than 16 bits are padded with 0 on the MSB side.

Expected response: RESULT_DATA.

| GET_SEL | Description | Number of GET_NEXT needed |
|---|---|---|
| 0x01 | Chip ID | 0 |
| 0x02 | HW version | 0 |
| 0x03 | Warm reset source (Fatal) | 0 |
| 0x04 | NVM CRC calculation | 0 |
| 0x05 | NVM CRC stored | 0 |
| 0x06 | SW version | 2 |
| 0x08 | Adder 2D | 0 |
| 0x09 | Adder 3D | 0 |
| 0x0A | Adder 4D | 1 |
| 0x10 | Raw 2D | 3 |
| 0x11 | Raw 3D | 4 |
| 0x12 | Raw 4D | 6 |
| 0x13 | Raw temp | 0 |
| 0x14 | Raw FDS (Full Diagnostic Sequence) | 7 |
| 0x15 | NV_DSP | 5 |

More details on the returned content are in the following sections.

The GET commands' main use are additional diagnostic functions that can be done on the user side.

### 5.4.1. GET, Chip ID

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x01 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x56 | | | | | | | |

This GET function returns the chip ID, which is stored inside the Melexis area of the IC. The RESULT_DATA contains the data as below.

```
DATA0 = CHIP_ID0[15:0]
DATA1 = CHIP_ID1[15:0]
DATA2 = CHIP_ID2[15:0]
```

### 5.4.2. GET, HW version

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x02 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x44 | | | | | | | |

This GET function returns the hardware version of the IC. This function can be used to identify the connected IC version. The RESULT_DATA contains the data as below.

```
DATA0 = DIG_VERSION[7:0] | ANA_VERSION[7:0]
DATA1 = DIG_VERSION[19:8]
```

The defaults are:

| | AAC |
|---|---|
| DATA0 | 0xAA4B |
| DATA1 | 0x0427 |

### 5.4.3. GET, Warm reset source

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x03 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x4A | | | | | | | |

This GET function returns information regarding the source of the warm reset that occurred. The RESULT_DATA contains the data as below.

DATA0 = RESET_CONTROLLER[15:0]
DATA1 = SOFT_RESET_STATUS[15:0]

With RESET_CONTROLLER[15:0]:

| Bit | Name | Description |
|---|---|---|
| 0 | DIAG_RAM_BIST | RAM BIST failing, HW reset |
| 1 | DIAG_ROM_BIST | ROM BIST failing, HW reset |
| 2 | DIAG_HW_ADDER (HW) | ADATA adder check failing, HW reset |
| 3 | DIAG_SYS_AWD | Absolute watchdog (AWD) HW reset |
| 4 | SOFT_WBOOT | SW reset (cfr SOFT_RESET_STATUS) |
| 5 | HVDIG_WBOOT | NVRAM supply failing, HW reset |
| 6 | DBG_WBOOT | Debugger HW reset |
| 7 | DIAG_SYS_TASK_ALIVENESS | Intelligent watchdog (IWD) HW reset |
| 8 | DIAG_CPU_STACKERR | CPU diagnostic, stack error |
| 9 | DIAG_CPU_PROTERR | CPU diagnostic, protection error |
| 10 | DIAG_CPU_MEMERR | CPU diagnostic, memory error |
| 11 | DIAG_CPU_OPERR | CPU diagnostic, operand error |
| 12 | DIAG_CPU_DMAERR | CPU diagnostic, DMA error |
| 13 | DIAG_RAM_PARITY | RAM parity diagnostics fault |
| 14 | DIAG_ROM_PARITY | ROM parity diagnostics fault |
| 15 | DIAG_NVM_ECC | NVRAM ECC diagnostics fault |

And SOFT_RESET_STATUS[15:0]:

| Bit | Name | Description |
|---|---|---|
| 0 | DIAG_ADC_CHECKSUM | ADC checksum diagnostic fault |
| 1 | DIAG_ADC_ERR_FATAL | ADC fatal error diagnostics fault |
| 2 | DIAG_HW_ADDER (SW) | ADATA adder check diagnostic fault |
| 3 | DIA_SYS_TASK_SEQ | Tasks sequence monitoring fault |
| 4 | DIAG_SYS_TASK_ALIVENESS | Idles task monitoring fault |
| 5 | DIAG_SYS_REG | Critical registers (IO ports) diagnostic fault |
| 6 | DIAG_DSP_ATAN2 | Arctangent processing monitoring fault |
| 7 | DIAG_DSP_COPRO | Co-processor monitoring fault |
| 8 | DIAG_SYS_NVM_STORE | NVM store command access verification (callback) fault |
| 9 | DIAG_SYS_MODE_CTRL | System modes state machine monitoring fault |
| 10 | DIAG_NVM_CRC_MLX | NVM CRC (Melexis page) verification fault |
| 11 | DIAG_NVM_CRC_USER | NVM CRC (user page) verification fault |
| 12 | CMD_RST | SPI reset command |
| 13 | CMD_RST_PARTIAL | SPI partial reset command |
| 14 | - | - |
| 15 | - | - |

### 5.4.4. GET, NVM CRC calculated

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x04 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x60 | | | | | | | |

This GET function returns the calculated NVRAM CRC for both user and Melexis area. The RESULT_DATA contains the data as below.

DATA0 = CRC_NVM_USER[15:0]
DATA1 = CRC_NVM_MLX[15:0]

The values can be compared to the stored CRC, and should equal.

### 5.4.5. GET, NVM CRC stored

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x05 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x6E | | | | | | | |

This GET function returns the stored NVRAM CRC for both user and Melexis area. The RESULT_DATA contains the data as below.

DATA0 = CRC_NVM_USER[15:0]
DATA1 = CRC_NVM_MLX[15:0]

The values can be compared to the calculated CRC, and should equal.

### 5.4.6. GET, SW version

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x06 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x7C | | | | | | | |

This GET function returns the software version of the IC. The RESULT_DATA contains the data as below.

DATA0 = MLX_GCC_VERSION[15:0]
DATA1 = MLX_GCC_VERSION[31:16]
DATA2 = MLX_PLTF_VERSION_MAJOR[7:0] | MLX_PLTF_VERSION_MINOR[7:0]
DATA0 = MLX_PLTF_VERSION_REVISION[7:0] | MLX_PLTF_VERSION_CUSTOMER_BUILD[7:0]
DATA1 = SW_TRIAXIS_VERSION_PRODUCT[7:0] | SW_TRIAXIS_VERSION_MAJOR[7:0]
DATA2 = SW_TRIAXIS_VERSION_MINOR[7:0] | SW_TRIAXIS_VERSION_REVISION[7:0]

The defaults are:

| | AAC |
|---|---|
| DATA0 | 0x0003 |
| DATA1 | 0x0178 |
| DATA2 | 0x0101 |
| DATA0 | 0x0E00 |
| DATA1 | 0x2703 |
| DATA2 | 0x0100 |
| DATA0 | N/A |
| DATA1 | N/A |

### 5.4.7. GET, Adder 2D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x08 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x28 | | | | | | | |

This GET function returns the ADATA adder outputs, differential mode only, for the latest acquisition and processing sequence for a 2-components sequence. The RESULT_DATA contains the data as below, which equal operations on the "Raw 2D" (section 5.4.10) data. Applicable after "legacy" or "dBz" acquisition.

| |
|---|
| DATA0 = B0DM[15:0] |
| DATA1 = B1DM[15:0] |

$$\mathrm{B0DM} = \frac{\mathrm{X0+X2\text{-}(X1+X3)}}{8},\ \mathrm{B1DM} = \frac{\mathrm{Y0+Y2\text{-}(Y1+Y3)}}{8}$$

### 5.4.8. GET, Adder 3D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x09 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x26 | | | | | | | |

This GET function returns the ADATA adder outputs, differential mode only, for the latest acquisition and processing sequence for a 3-components sequence. The RESULT_DATA contains the data as below, which equal operations on the "Raw 3D" (section 5.4.11) data. Applicable after "joystick" or "3D-fields" acquisition.

| |
|---|
| DATA0 = B0DM[15:0] |
| DATA1 = B1DM[15:0] |
| DATA2 = B2DM[15:0] |

$$\mathrm{B0DM} = \frac{\mathrm{X0+X2\text{-}(X1+X3)}}{8},\ \mathrm{B1DM} = \frac{\mathrm{Y0+Y2\text{-}(Y1+Y3)}}{8},\ \mathrm{B2DM} = \frac{\mathrm{Z0+Z2\text{-}(Z1+Z3)}}{8}$$

### 5.4.9. GET, Adder 4D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x0A | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x34 | | | | | | | |

This GET function returns the ADATA adder outputs, differential mode only, for the latest acquisition and processing sequence for a 4-components sequence. The RESULT_DATA contains the data as below, which equal operations on the "Raw 4D" (section 5.4.12) data. Applicable after "dual" acquisition.

DATA0 = B0DM[15:0]
DATA1 = B1DM[15:0]
DATA2 = 0x0000
DATA0 = B2DM[15:0]
DATA1 = B3DM[15:0]

$$\mathrm{B0DM} = \frac{\mathrm{XA0+XA2\text{-}(XA1+XA3)}}{8},\ \mathrm{B1DM} = \frac{\mathrm{YA0+YA2\text{-}(YA1+YA3)}}{8},\ \mathrm{B2DM} = \frac{\mathrm{XB0+XB2\text{-}(XB1+XB3)}}{8},\ \mathrm{B3DM} = \frac{\mathrm{YB0+YB2\text{-}(YB1+YB3)}}{8}$$

### 5.4.10. GET, Raw 2D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x10 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0xB8 | | | | | | | |

This GET function returns the raw data (4-phase spinning) for the latest acquisition and processing sequence for a 2-components sequence. The RESULT_DATA contains the data as below. Applicable after “legacy” or “dBz” acquisition.

| |
|---|
| DATA0 = X0[14:0] |
| DATA1 = X1[14:0] |
| DATA2 = Y0[14:0] |
| DATA0 = Y1[14:0] |
| DATA1 = X2[14:0] |
| DATA2 = X3[14:0] |
| DATA0 = Y2[14:0] |
| DATA1 = Y3[14:0] |
| DATA2 = T[13:0][4] |
| DATA0 = HW_ADC_CHECKSUM[15:0] |
| DATA1 = PARTIAL_SW_ADC_CHECKSUM[15:0] |

Test performed:

$$\mathrm{HW\_ADC\_CHECKSUM} = \mathrm{PARTIAL\_SW\_ADC\_CHECKSUM} + \sum_{i=0}^{3} Xi + \sum_{i=0}^{3} Yi$$

[4] *The values here are equal to those received in section 5.4.13 (GET, Raw temperature).*

### 5.4.11. GET, Raw 3D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| 0x00[1] | | | | | | | | 0x00[1] | | | | | | | |
| GET_SEL[7:0] = 0x11 | | | | | | | | 0x00[1] | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0xB6 | | | | | | | |

This GET function returns the raw data (4-phase spinning) for the latest acquisition and processing sequence for a 3-components sequence. The RESULT_DATA contains the data as below. Applicable after “joystick” or “3D-fields” acquisition.

| |
|---|
| DATA0 = X0[14:0]<br>DATA1 = X1[14:0]<br>DATA2 = Y0[14:0]<br>DATA0 = Y1[14:0]<br>DATA1 = Z0[14:0]<br>DATA2 = Z1[14:0]<br>DATA0 = X2[14:0]<br>DATA1 = X3[14:0]<br>DATA2 = Y2[14:0]<br>DATA0 = Y3[14:0]<br>DATA1 = Z2[14:0]<br>DATA2 = Z3[14:0]<br>DATA0 = T[13:0][4]<br>DATA1 = HW_ADC_CHECKSUM[15:0]<br>DATA2 = PARTIAL_SW_ADC_CHECKSUM[15:0] |

Test performed:

$$\mathrm{HW\_ADC\_CHECKSUM} = \mathrm{PARTIAL\_SW\_ADC\_CHECKSUM} + \sum_{i=0}^{3} \mathrm{X}i + \sum_{i=0}^{3} \mathrm{Y}i + \sum_{i=0}^{3} \mathrm{Z}i$$

### 5.4.12. GET, Raw 4D

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x12 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0xA4 | | | | | | | |

This GET function returns the raw data (4-phase spinning) for the latest acquisition and processing sequence for a 4-components sequence. The RESULT_DATA contains the data as below. Applicable after "dual" acquisition.

```
DATA0 = XA0[14:0]
DATA1 = XA1[14:0]
DATA2 = YA0[14:0]
DATA0 = YA1[14:0]
DATA1 = XA2[14:0]
DATA2 = XA3[14:0]
DATA0 = YA2[14:0]
DATA1 = YA3[14:0]
DATA2 = XB0[14:0]
DATA0 = XB1[14:0]
DATA1 = YB0[14:0]
DATA2 = YB1[14:0]
DATA0 = XB2[14:0]
DATA1 = XB3[14:0]
DATA2 = YB2[14:0]
DATA0 = YB3[14:0]
DATA1 = T[13:0](4)
DATA2 = HW_ADC_CHECKSUM[15:0]
DATA0 = PARTIAL_SW_ADC_CHECKSUM[15:0]
```

Test performed:

$$\text{HW\_ADC\_CHECKSUM} = \text{PARTIAL\_SW\_ADC\_CHECKSUM} + \sum_{i=0}^{3} \text{XA}i + \sum_{i=0}^{3} \text{YA}i + \sum_{i=0}^{3} \text{XB}i + \sum_{i=0}^{3} \text{YB}i$$

### 5.4.13. GET, Raw temperature

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x13 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0xAA | | | | | | | |

This GET function returns the raw data for temperature and diagnostic temperature. The RESULT_DATA contains the data as below. These are raw values before linearization (so not the same as returned in RESULT_MEAS).

DATA0 = T[13:0]
DATA1 = T_DIAG[13:0]

The formula to go from this raw temperature to the temperature returned in RESULT_DATA is shown below:

$$T_{LIN} = 864 + \frac{(13107 + T_{LINSLOPE} \cdot 2^7)}{2^{16}} \cdot \left(T_{RAW} - (7000 + T_{RAWREF})\right)$$

In this formula, the $T_{LINSLOPE}$ value depends on the sign of $\left(T_{RAW} - (7000 + T_{RAWREF})\right)$. If positive, the $T_{LINSLOPEHOT}$ is taken, else the $T_{LINSLOPECOLD}$ is taken. These slopes, together with $T_{RAWREF}$ are found in the NVRAM of the IC at the locations according the table below:

| | AAC |
|---|---|
| $T_{RAWREF}$ | 0x109A[11:0] |
| $T_{LINSLOPECOLD}$ | 0x1098[7:0] |
| $T_{LINSLOPEHOT}$ | 0x1098[15:8] |
| NV_TRIM_DIAG_LOW_TEMP_THRES | 0x1070[7:0] |
| NV_TRIM_DIAG_HIGH_TEMP_THRES | 0x1070[15:8] |

$T_{LIN}$ is clamped between 1 and 4088, or if NV_DSP_TEMP_CLIP_EN=1 clamped between NV_TRIM_DIAG_LOW_TEMP_THRES<<4 and NV_TRIM_DIAG_HIGH_TEMP_THRES<<4. The defaults for this latter two parameters are 0x08 (-57.2°C) and 0x88 (198.8°C) respectively. These two values are also used for the over and under temperature diagnostics.

### 5.4.14. GET, Raw FDS (Full Diagnostic Sequence)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00[(1)] | | | | | | | | 0x00[(1)] | | | | | | | |
| 0x00[(1)] | | | | | | | | 0x00[(1)] | | | | | | | |
| GET_SEL[7:0] = 0x14 | | | | | | | | 0x00[(1)] | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x80 | | | | | | | |

This GET function returns the raw ADC values for the full diagnostic sequence. These values are only available after the execution of a full diagnostic acquisition sequence. The RESULT_DATA contains the data as below, and is shown in the table together with the description and the limits for the test in case applicable. **Note that the limits are taken on the half of the value for both ADC reference points as calculations on test bridge points (so the returned values can be seen as doubled with respect to the Min/Typ/Max values)**.

| | Description | Min | Typ | Max |
|---|---|---|---|---|
| DATA0 = T | Temperature[(5)] | | | |
| DATA1 = D0 | Diagnostic temperature (temp sensor 2)[(5)] | | | |
| DATA2 = D1 | ADC self-test reference point 0 | 0x017F | 0x01FF | 0x027F |
| DATA0 = D2 | ADC self-test reference point 8 | 0x237F | 0x23FF | 0x247F |
| DATA1 = D3 | ADC self-test reference point 4 | 0x117F | 0x11FF | 0x127F |
| DATA2 = D4 | ADC self-test reference point 12 | 0x337F | 0x33FF | 0x347F |
| DATA0 = D5 | ADC self-test reference point 1 | 0x037F | 0x03FF | 0x047F |
| DATA1 = D6 | ADC self-test reference point 9 | 0x257F | 0x25FF | 0x267F |
| DATA2 = D7 | ADC self-test reference point 5 | 0x137F | 0x13FF | 0x147F |
| DATA0 = D8 | ADC self-test reference point 13 | 0x357F | 0x35FF | 0x367F |
| DATA1 = D9 | ADC self-test reference point 2 | 0x097F | 0x09FF | 0x0A7F |
| DATA2 = D10 | ADC self-test reference point 10 | 0x2B7F | 0x2BFF | 0x2C7F |
| DATA0 = D11 | ADC self-test reference point 6 | 0x197F | 0x19FF | 0x1A7F |
| DATA1 = D12 | ADC self-test reference point 14 | 0x3B7F | 0x3BFF | 0x3C7F |
| DATA2 = D13 | ADC self-test reference point 3 | 0x0B7F | 0x0BFF | 0x0C7F |
| DATA0 = D14 | ADC self-test reference point 11 | 0x2D7F | 0x2DFF | 0x2E7F |
| DATA1 = D15 | ADC self-test reference point 7 | 0x1B7F | 0x1BFF | 0x1C7F |
| DATA2 = D16 | ADC self-test reference point 15 | 0x3D7F | 0x3DFF | 0x3E7F |
| DATA0 = D17 | Test bridge reference point 0 | | | |
| DATA1 = D18 | Test bridge reference point 1 | | | |
| DATA2 = D19 | Test bridge reference point 2 | | | |
| DATA0 = D20 | Test bridge reference point 3 | | | |
| | Differential mode, narrow: D18-D17 | 4700 | 5050 | 5400 |
| | Common mode, narrow: D18+D17 | 14484 | 16384 | 18284 |
| | Differential mode, wide: D20-D19 | 10700 | 11350 | 12000 |
| | Common mode, wide: D20+D19 | 14384 | 16384 | 18384 |

[5] *The values here are doubled those received in section 5.4.13 (GET, Raw temperature).*

### 5.4.15. GET, NV DSP

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| 0x00(1) | | | | | | | | 0x00(1) | | | | | | | |
| GET_SEL[7:0] = 0x15 | | | | | | | | 0x00(1) | | | | | | | |
| 0 | 0x07 (GET) | | | | | | | CRC[7:0] = 0x8E | | | | | | | |

This GET function returns all relevant NVRAM parameters to process raw data and calculate the angle. The RESULT_DATA contains the data as below.

DATA0 = NV_DSP_GAIN[7:0] | NV_DSP_REVPOL | NV_DSP_FILTER[1:0] | NV_DSP_TEMP_CLIP_EN
DATA1 = NV_DSP_DP[13:0]
DATA2 = NV_DSP_CLAMPLOW[11:0]
DATA0 = NV_DSP_CLAMPHIGH[11:0]
DATA1 = NV_DSP_OFS_X_TMID_LEGACY[15:0]
DATA2 = NV_DSP_OFS_Y_TMID_LEGACY[15:0]
DATA0 = NV_DSP_OFS_X_TMID_HALFHP[15:0]
DATA1 = NV_DSP_OFS_Y_TMID_HALFHP[15:0]
DATA2 = NV_DSP_OFS_Y_TC_HOT_LEGACY[7:0] | NV_DSP_OFS_X_TC_HOT_LEGACY[7:0]
DATA0 = NV_DSP_OFS_Y_TC_COLD_LEGACY[7:0] | NV_DSP_OFS_X_TC_COLD_LEGACY[7:0]
DATA1 = NV_DSP_TLIN_SLOPE_HOT[7:0] | NV_DSP_TLIN_SLOPE_COLD[7:0]
DATA2 = NV_DSP_TSENSOR_ADC_REF[11:0]
DATA0 = NV_DSP_S_XX_LEGACY[15:0]
DATA1 = NV_DSP_S_YY_LEGACY[15:0]
DATA2 = NV_DSP_S_XY_LEGACY[15:0]
DATA0 = NV_DSP_FINEOFFSET_X4 | NV_DSP_FINEOFFSET_X2 | NV_DSP_SCOMP_HALFHP_EN[1:0]
DATA1 = NV_TRIM_TMID[3:0]

## 5.5. SET Command

There is one more type of commands, those set information, data, functions, or parameters from the IC. The SET command looks like below

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DATA0[15:0] | | | | | | | | | | | | | | | |
| DATA1[15:0] | | | | | | | | | | | | | | | |
| SET_SEL[7:0] | | | | | | | | 0x00[1] | | | | | | | |
| 0 | 0x0D (SET) | | | | | | | CRC[7:0] | | | | | | | |

Sends a SET command to the IC. Depending on **SET_SEL**, a different **DATA0**/**DATA1**/**DATA2** content can be written. DATA containing values shorter than 16 bits are padded with 0 on the MSB side. Since there is only one 2-word option to set.

| SET_SEL | Description | RESULT_DATA content (color change indicates GET_NEXT required) |
|---|---|---|
| 0x01 | MWD[6] | DATA0 = MWD_MIN[13:0]<br>DATA1 = MWD_MAX[13:0] |

MWD: this command is used to configure, start and feed the watchdog, with the parameters each having a resolution of 100µs/LSB. MWD_MIN is the window opening time, and MWD_MAX the window closing time (thus counter timeout). If MWD_MIN=MWD_MAX=0, then the MWD will not be started or will be stopped if running previously. A MWD_MIN larger or equal to MWD_MAX (and different from 0) results in the same action but also an error (ERR_ARGS) will be thrown.

Expected response: RESULT_ACK.

[6] *MWD = Master Watch Dog.*

# 6. MISO Messages

## 6.1. Measurement Results

### 6.1.1. RESULT_MEAS

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>S1</td><td>S0</td><td colspan="14">DATA_PRIMARY[13:0]</td></tr>
<tr><td colspan="16">DATA_SECONDARY[15:0]</td></tr>
<tr><td colspan="4">MODE[3:0]</td><td colspan="12">TEMP[11:0]</td></tr>
<tr><td>1</td><td>0</td><td colspan="6">MEAS_COUNT[5:0]</td><td colspan="8">CRC[7:0]</td></tr>
</table>

The measurement result is a reply to the normal trigger or synchronous trigger. The **DATA_PRIMARY** and **DATA_SECONDARY** are according what was requested, and the **MODE** is confirmed too. A measurement counter, **MEAS_COUNT**, is included (0x01 to 0x3F, rolling back to 0x01 after 0x3F) to detect missing measurements. Temperature information is included in **TEMP** and encoded according the below formula:

$$Temperature(°C) = \frac{TEMP[11:0]}{8} - 73.15$$

**S1** and **S0** are status flags, encoded as in the table below:

| S1 | S0 | Description / Validity of Measurements |
|---|---|---|
| 0 | 0 | No errors.<br>Valid measurements returned. |
| 0 | 1 | Warning, cycle time violation.<br>Measurements may not be valid (safe state 4). |
| 1 | 0 | Error detected, diagnostic fail.<br>Measurement shall not be considered as valid (safe state 2, safe state 3). |
| 1 | 1 | Warning and error condition at the same time.<br>Measurement shall not be considered as valid. |

### 6.1.2. RESULT_MEAS_3D

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>S1</td><td>S0</td><td colspan="14">FIELD_B0[13:0]</td></tr>
<tr><td>0</td><td>0</td><td colspan="14">FIELD_B1[13:0]</td></tr>
<tr><td>1</td><td>1</td><td colspan="14">FIELD_B2[13:0]</td></tr>
<tr><td>1</td><td>0</td><td colspan="6">MEAS_COUNT[5:0]</td><td colspan="8">CRC[7:0]</td></tr>
</table>

This message is returned for the case of the mode equal to “Fields 3D”. **FIELD_B0/B1/B2** holds the measurement data for the three components. The **S1** and **S0** are status flags, as in the RESULT_MEAS message, same is valid for the measurement counter **MEAS_COUNT**.

## 6.2. Returned Data

### 6.2.1. RESULT_DATA

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DATA0[15:0] | | | | | | | | | | | | | | | |
| DATA1[15:0] | | | | | | | | | | | | | | | |
| DATA2[15:0] | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | FRAME_COUNT[4:0] | | | | | CRC[7:0] | | | | | | | |

When data is requested from the IC, which are not measurements, this is the format it will be returned in. The data is held in **DATA0/DATA1/DATA2**. A frame counter, **FRAME_COUNT** (starting at 0x00), is available to help identifying the contents in case of a burst NVRAM read (with READ_NEXT) or in case of GET_NEXT commands.

## 6.3. Status, Acknowledge, Error

### 6.3.1. RESULT_STATUS

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">DIAGS_STATE[31:16]</td></tr>
<tr><td colspan="16">DIAGS_STATE[15:0]</td></tr>
<tr><td colspan="4">TYPE[3:0] = 0x0</td><td colspan="4">STATE[3:0]</td><td colspan="8">0x00</td></tr>
<tr><td>0</td><td colspan="7">OPC[6:0]</td><td colspan="8">CRC[7:0]</td></tr>
</table>

A response of the type “Result Status” has always the **TYPE** field set to 0x0. The **STATE** the sensor is in is reported according to the table below.

| STATE | Description |
|---|---|
| 0x0 | Startup |
| 0x1 | Safe startup |
| 0x4 | Normal |
| 0x5 | Protected mode |

The decoding of the diagnostic bits (**DIAGS_STATE**) is shown in the section 6.4. A RESULT_STATUS is expected after a NOP or a TRIG_NORMAL/SYNC with the full diagnostic sequence option set. The message also shows the opcode (**OPC**) of the message that “triggered” the reply.

### 6.3.2. RESULT_ACK

The returned messages show the opcode (OPC) of the message that “caused” the reply.

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">0x00</td><td colspan="8">0x00</td></tr>
<tr><td colspan="8">0x00</td><td colspan="8">0x00</td></tr>
<tr><td colspan="4">TYPE[3:0] = 0x1</td><td colspan="4">0x0</td><td colspan="8">FRAME_COUNT[7:0]</td></tr>
<tr><td>0</td><td colspan="7">OPC[6:0]</td><td colspan="8">CRC[7:0]</td></tr>
</table>

A response of the type “Result ACK” has always the **TYPE** field set to 0x1. A frame counter, **FRAME_COUNT** (starting at 0x00), is included. A RESULT_ACK is expected after SET/SET_NEXT, PROTECTED_MODE, EXIT, NVM_RECALL, NVM_STORE, and WRITE/WRITE_NEXT. The message also shows the opcode (**OPC**) of the message that “triggered” the reply.

### 6.3.3. ERROR

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">DIAGS_STATE[31:16] (ERR_DIAGS) or 0x0000</td></tr>
<tr><td colspan="16">DIAGS_STATE[15:0] (ERR_DIAGS) or 0x0000</td></tr>
<tr><td colspan="4">TYPE[3:0] = 0x8</td><td colspan="4">0x0</td><td colspan="8">ERROR_CODE[7:0]</td></tr>
<tr><td>0</td><td colspan="7">OPC[6:0]</td><td colspan="8">CRC[7:0]</td></tr>
</table>

In case of an error, this message is returned. The **TYPE** field equals 0x8. The table below shows the possible **ERROR_CODE**s, ranked by priority (ERR_FRAME has top priority). The ERROR message has always priority over other messages. The message also shows the opcode (**OPC**) of the message that “triggered” the reply, and the **DIAGS_STATE** (section 6.4) in case of ERROR_CODE=0x0F.

| **ERROR_CODE** | **Description** |
|---|---|
| 0xCC | **ERR_FRAME**, low level error (CS rising edge received in middle of a byte, RX/TX buffer overflow). |
| 0x69 | **ERR_CRC**, a CRC error on the MOSI message is detected. |
| 0x33 | **ERR_RDY**, system is not yet ready to accept and process any command (during startup or safe startup phases). |
| 0x5A | **ERR_ONGOING**, previous command is still ongoing (includes opcode) and any other command is being refused for now. Currently received command is dropped and result of ongoing command is delayed to next transaction. |
| 0x3C | **ERR_OPC**, an invalid opcode is received. |
| 0x55 | **ERR_STATE**, a command was received that is not accepted in the current state. |
| 0x96 | **ERR_KEY**, an invalid key is received. |
| 0x66 | **ERR_ACCESS**, the current access level is not accepting the command. |
| 0x99 | **ERR_ADDRESS**, an invalid or byte-aligned address is provided to read from or write to. |
| 0xA5 | **ERR_ARGS**, invalid arguments are provided. |
| 0xAA | **ERR_TIME**, a timeout is detected, on for example the measurements. |
| 0x0F | **ERR_DIAGS**, a diagnostic error is detected, with **DIAGS_STATE** returned in bytes 7 to 4. The decoding of the diagnostic bits is shown in the section 6.4. |
| 0xC3 | **ERR_STORE**, the store operation is faulty (not performed). This can be due to under-voltage detected, wrong CRC or already locked NVRAM. |

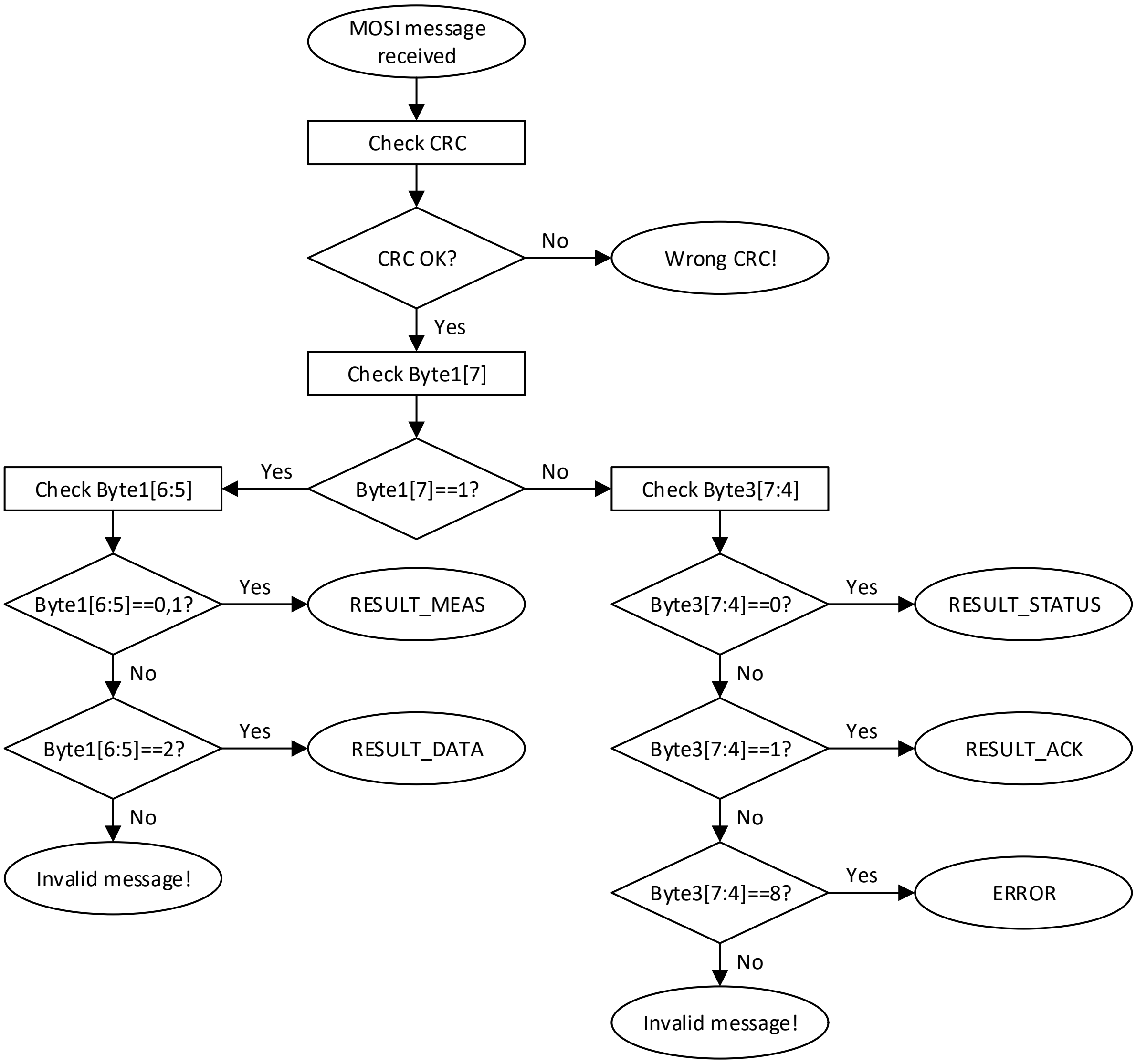
### 6.4. Diagnostics

The DIAGS_STATE bytes are encoded following the table below.

| Bit | Name | Description |
|---|---|---|
| 0 | ADC_ERR | ADC error detected. |
| 1 | SYS_ADC_TIME | ADC sequence timing (TRG-to-EOS) fail detected. |
| 2 | SYS_APS_TIME | APS sequence timing (TRG-to-End of DSP) fail detected. |
| 3 | DSP_OVF_APS | DSP overflow in APS (prevention & detection). |
| 4 | OV_VDD_5V | VDD over-voltage detected. |
| 5 | UV_VDD_5V | VDD under-voltage detected. |
| 6 | OV_VDDA | VDDA over-voltage detected. |
| 7 | UV_VDDA | VDDA under-voltage detected. |
| 8 | OV_VDDD | VDDD over-voltage detected. |
| 9 | AFE_HP_DIAG | Diagnostic rotary magnetic mode comparison check failed. |
| 10 | AFE_HP_DUAL | Dual mode rotary magnetic mode comparison check failed. |
| 11 | AFE_AROC | Automatic rough offset correction (AROC) consistency failed. |
| 12 | AFE_GAIN | Automatic gain control (AGC) consistency failed. |
| 13 | AFE_FIELD_MAG_HIGH | Field magnitude too high. |
| 14 | AFE_FIELD_MAG_LOW | Field magnitude too low. |
| 15 | DSP_OVF_BTF | DSP overflow in background tasks (prevention & detection). |
| 16 | HIGH_TEMP | Over-temperature detected. |
| 17 | LOW_TEMP | Under-temperature detected. |
| 18 | ADC_REF | ADC self-test (reference test points) failed. |
| 19 | AFE_TEMP | Temperature sensors comparison failed. |
| 20 | AFE_TESTBRIDGE | AFE test bridge diagnostic (reference test points) failed. |
| 21 | - | Always 0. |
| 22 | SYS_CTM_LEGACY | Warning, cycle time monitoring for AGC and AROC calculation for legacy acquisition failed. |
| 23 | SYS_CTM_DCZ | Warning, cycle time monitoring for AGC and AROC calculation for dBz acquisition failed. |
| 24 | SYS_CTM_TEMP | Warning, cycle time monitoring for temperature compensation calculation failed. |
| 25 | SYS_DCT | Warning, diagnostic cycle time monitoring failed. |
| 26 | - | Always 0. |
| 27 | - | Always 0. |
| 28 | - | Always 0. |
| 29 | - | Always 0. |
| 30 | - | Always 0. |
| 31 | - | Always 0. |

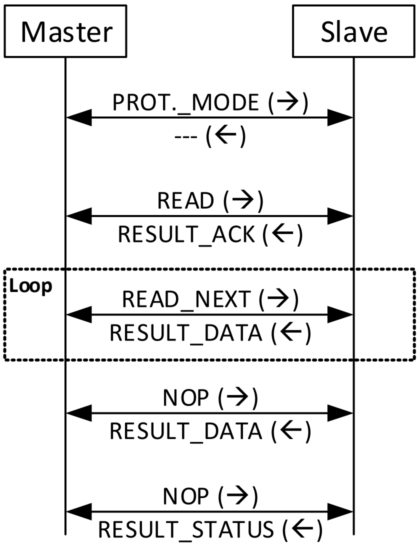
### 6.5. MISO Message Decoding Strategy

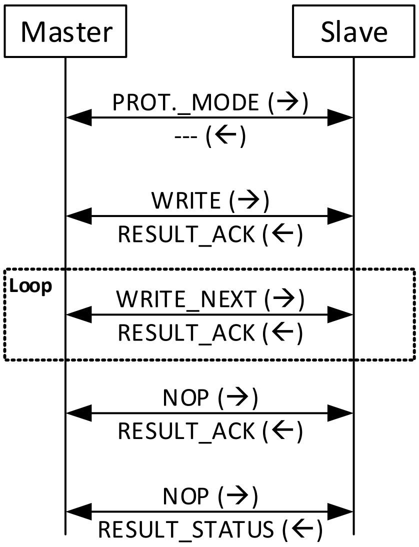
To decode the MISO message, the below strategy can be followed to classify the received bytes.



# 7. Reference Sequences

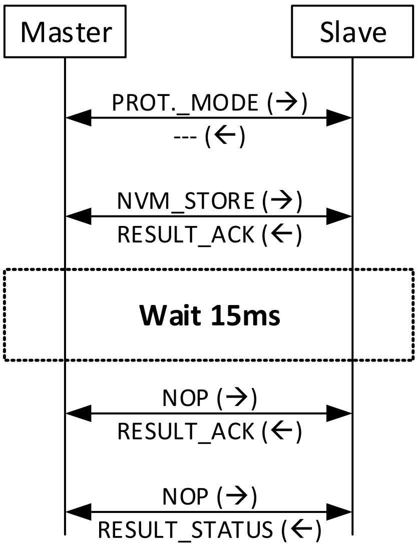
## 7.1. Reading/Writing NVRAM

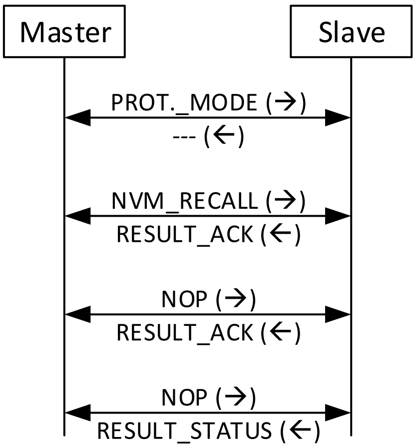



Note: Make sure to set the NVM CRC-16 correct, if changing NVRAM content. In case it is wrong, it will be detected by the diagnostics mechanisms and the IC will perform a reset.

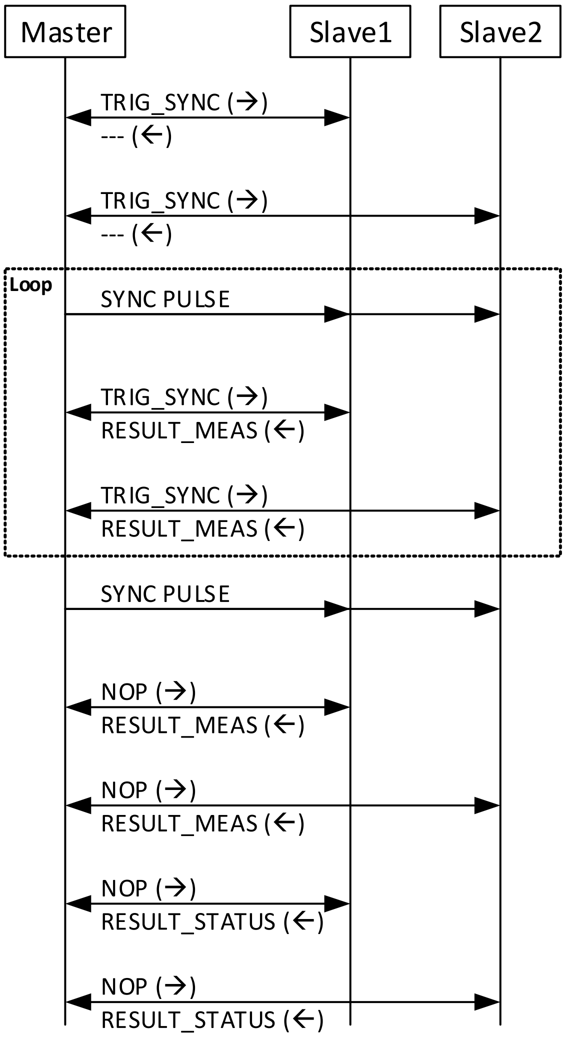
## 7.2. Store/Recall NVRAM

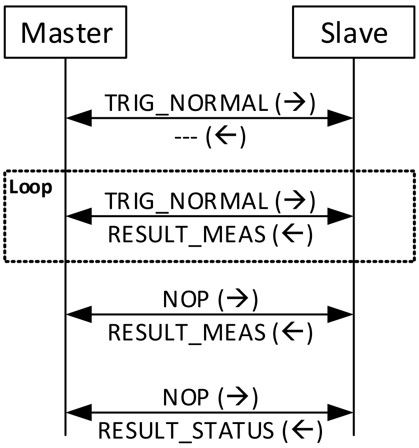



Note: For the store operation to succeed, make sure to have the NVM CRC-16 correct. The supply needs to be stable during the 15ms wait.

## 7.3. Normal/Sync Measurements



## 8. Revision History

| Revision | Date | Changes |
|---|---|---|
| 001 | March 25, 2024 | Creation |
| | | |

## 9. Disclaimer

*The content of this document is believed to be correct and accurate. However, the content of this document is furnished "as is" for informational use only and no representation, nor warranty is provided by Melexis about its accuracy, nor about the results of its implementation. Melexis assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. Customer will follow the practices contained in this document under its sole responsibility. This documentation is in fact provided without warranty, term, or condition of any kind, either implied or expressed, including but not limited to warranties of merchantability, satisfactory quality, non-infringement, and fitness for purpose. Melexis, its employees and agents and its affiliates' and their employees and agents will not be responsible for any loss, however arising, from the use of, or reliance on this document. Notwithstanding the foregoing, contractual obligations expressly undertaken in writing by Melexis prevail over this disclaimer.*

*This document is subject to change without notice, and should not be construed as a commitment by Melexis. Therefore, before placing orders or prior to designing the product into a system, users or any third party should obtain the latest version of the relevant information. Users or any third party must determine the suitability of the product described in this document for its application, including the level of reliability required and determine whether it is fit for a particular purpose.*

*This document as well as the product here described may be subject to export control regulations. Be aware that export might require a prior authorization from competent authorities. The product is not designed, authorized or warranted to be suitable in applications requiring extended temperature range and/or unusual environmental requirements. High reliability applications, such as medical life-support or life-sustaining equipment or avionics application are specifically excluded by Melexis. The product may not be used for the following applications subject to export control regulations: the development, production, processing, operation, maintenance, storage, recognition or proliferation of:*

*1. chemical, biological or nuclear weapons, or for the development, production, maintenance or storage of missiles for such weapons;*
*2. civil firearms, including spare parts or ammunition for such arms;*
*3. defense related products, or other material for military use or for law enforcement;*
*4. any applications that, alone or in combination with other goods, substances or organisms could cause serious harm to persons or goods and that can be used as a means of violence in an armed conflict or any similar violent situation.*

*No license nor any other right or interest is granted to any of Melexis' or third party's intellectual property rights.*

*If this document is marked "restricted" or with similar words, or if in any case the content of this document is to be reasonably understood as being confidential, the recipient of this document shall not communicate, nor disclose to any third party, any part of the document without Melexis' express written consent. The recipient shall take all necessary measures to apply and preserve the confidential character of the document. In particular, the recipient shall (i) hold document in confidence with at least the same degree of care by which it maintains the confidentiality of its own proprietary and confidential information, but no less than reasonable care; (ii) restrict the disclosure of the document solely to its employees for the purpose for which this document was received, on a strictly need to know basis and providing that such persons to whom the document is disclosed are bound by confidentiality terms substantially similar to those in this disclaimer; (iii) use the document only in connection with the purpose for which this document was received, and reproduce document only to the extent necessary for such purposes; (iv) not use the document for commercial purposes or to the detriment of Melexis or its customers. The confidentiality obligations set forth in this disclaimer will have indefinite duration and in any case they will be effective for no less than 10 years from the receipt of this document.*

*This disclaimer will be governed by and construed in accordance with Belgian law and any disputes relating to this disclaimer will be subject to the exclusive jurisdiction of the courts of Brussels, Belgium.*

*The invalidity or ineffectiveness of any of the provisions of this disclaimer does not affect the validity or effectiveness of the other provisions.*
*The previous versions of this document are repealed.*



*IATF 16949 and ISO 14001 Certified*