ICM-42607x and ICM-42670x DMP Mode Accelerometer and Gyroscope Self-Test

InvenSense, a TDK Group Company
1745 Technology Drive, San Jose, CA 95110 U.S.A
+1(408) 988–7339
invensense.tdk.com

Document Number: AN-000258
Revision: 1.1
Date: 03/23/2021

Table of Contents

1 SELF-TEST OVERVIEW

This document explains how to run the DMP-based self-test procedure in ICM-42607x and ICM-42670x and the associated parameters.

The ICM-42607x and ICM-42670x self-test is embedded in DMP ROM software and enables customers to perform a functional test of the mechanical and electrical integrity of the sensor without requiring physical device movement.

When running self-test, DMP is responsible for accel and gyro configuration. It is not required for the host to save/restore sensor configuration.

'During self-test, the part under test must be static and without movement. Any motion applied to the part under test will cause self-test to fail.

Self-test uses the same RAM section as the APEX features. Therefore, the host has to re-initialize APEX after self-test execution.

2 REGISTER FIELDS FOR SELF-TEST OPERATION

This section summarizes all bitfields and registers required to properly operate self-tests from the host.

2.1 SELF-TEST PARAMETERS

The register fields below are used to configure self-test.

BIT FIELD	BANK AND ADDRESS	REGISTER NAME	VALUE (HEX) FOR ST	FUNCTIONALITY
ACCEL_ST_LIM	Bank MREG1 Register 0x13 Bit [5:3]	ST_CONFIG	7	These bits control the tolerated ratio between self-test processed values and reference (fused) ones for accelerometer 7: 50% ~150%
GYRO_ST_LIM	Bank MREG1 Register 0x13 Bit [2:0]	ST_CONFIG	7	These bits control the tolerated ratio between self-test processed values and reference (fused) ones for gyroscope 7: 50% ~150%
ST_NUMBER_SAMPLE	Bank MREG1 Register 0x13 Bit [6]	ST_CONFIG	0 or 1	Selects the number of samples used for self-test. 0: 16 samples 1: 200 samples
ACCEL_ST_EN	Bank MREG1 Register 0x14 Bit [6]	SELFTEST	1	1: Enable accel self-test operation To execute self-test for both accel and gyro, accel_st_en and gyro_st_en should be set in the same write access.
GYRO_ST_EN	Bank MREG1 Register 0x14 Bit [7]	SELFTEST	1	1: Enable gyro self-test operation To execute self-test for both accel and gyro, accel_st_en and gyro_st_en should be set in the same write access.

2.2 CONTROL REGISTERS

The following register fields are required for self-test operation.

BIT FIELD	BANK AND ADDRESS	REGISTER NAME	VALUE (HEX) FOR ST	FUNCTIONALITY
GYRO_MODE	Bank 0 Register 0x1F Bit [3:2]	PWR_MGMT0	0	Controls gyro sensor. Will need to be turned off before executing self-test. 0: OFF
ACCEL_MODE	Bank 0 Register 0x1F Bit [1:0]	PWR_MGMT0	0	Controls accel sensor. Will need to be turned off before executing self-test. 0: OFF
IDLE	Bank 0 Register 0x1F Bit [4]	PWR_MGMT0	1	1: the RC oscillator is powered on even if Accel and Gyro are powered off. This bit enables MCLK.
MCLK_RDY	Bank 0 Register 0x00 Bit [3]	MCLK_RDY	Wait until 1	0: Indicates internal clock is currently not running 1: Indicates internal clock is currently running
DMP_MEM_RESET_EN	Bank 0 Register 0x25 Bit [0]	APEX_CONFIG0	1	When this bit is set to 1, it clears DMP SRAM for APEX operation or Self-test operation.
OTP_COPY_MODE	Bank MREG1 Register 0x2B Bit [3:2]	OTP_CONFIG	3	11: Enable copying self-test data from OTP memory to SRAM
OTP_PWR_DOWN	Bank MREG2 Register 0x06 Bit [1]	OTP_CTRL7	0	0: Power up OTP to copy from OTP to SRAM 1: Power down OTP This bit is automatically set to 1 when OTP copy operation is complete.
OTP_RELOAD	Bank MREG2 Register 0x06 Bit [3]	OTP_CTRL7	1	Setting this bit to 1 triggers OTP copy operation.
ST_INT1_EN	Bank 0 Register 0x2B Bit [7]	INT_SOURCE0	1	0: Self-Test Done interrupt not routed to INT1 1: Self-Test Done interrupt routed to INT1
ST_INT	Bank 0 Register 0x3A Bit [7]	INT_STATUS	Status	This bit automatically sets to 1 when a Self Test done interrupt is generated. The bit clears to 0 after the register has been read.
WHOAMI	Bank 0 Register 0x75 Bit [7:0]	WHO_AM_I	ID	Register to indicate to user which device is being accessed.

2.3 RESULTS REGISTERS

The outcome of the self-test routine will be available in the following register fields.

BIT FIELD	BANK AND ADDRESS	REGISTER NAME	FUNCTIONALITY
ACCEL_ST_PASS	Bank MREG1 Register 0x63 Bit [5]	ST_STATUS1	1: Accel self-test passed for all the 3 axes
ACCEL_ST_DONE	Bank MREG1 Register 0x63 Bit [4]	ST_STATUS1	1: Accel self-test done for all the 3 axes
AZ_ST_PASS	Bank MREG1 Register 0x63 Bit [3]	ST_STATUS1	1: Accel Z-axis self-test passed
AY_ST_PASS	Bank MREG1 Register 0x63 Bit [2]	ST_STATUS1	1: Accel Y-axis self-test passed
AX_ST_PASS	Bank MREG1 Register 0x63 Bit [1]	ST_STATUS1	1: Accel X-axis self-test passed
ST_INCOMPLETE	Bank MREG1 Register 0x64 Bit [6]	ST_STATUS2	1: Self-test is incomplete. This bit is set to 1 if the self-test was aborted. One possible cause of aborting the self-test may be the detection of significant movement in the gyro when the self-test for gyro and/or accel is being executed.
GYRO_ST_PASS	Bank MREG1 Register 0x64 Bit [5]	ST_STATUS2	1: Gyro self-test passed for all the 3 axes
GYRO_ST_DONE	Bank MREG1 Register 0x64 Bit [4]	ST_STATUS2	1: Gyro self-test done for all the 3 axes
GZ_ST_PASS	Bank MREG1 Register 0x64 Bit [3]	ST_STATUS2	1: Gyro Z-axis self-test passed
GY_ST_PASS	Bank MREG1 Register 0x64 Bit [2]	ST_STATUS2	1: Gyro Y-axis self-test passed
GX_ST_PASS	Bank MREG1 Register 0x64 Bit [1]	ST_STATUS2	1: Gyro X-axis self-test passed

3 SELF-TEST PROCEDURE

While the self-test operation is running (either ACCEL_ST_EN or GYRO_ST_EN is set to 1), the host should not perform any write operation to registers.

To execute self-test, follow the routine shown below:

```
# Disables Gyro/Accel sensors
BANK0.PWR_MGMT0.gyro_mode = 0
BANK0.PWR_MGMT0.accel_mode = 0

# Enable RC oscillator
BANK0.PWR_MGMT0.idle = 1

# Clear DMP SRAM
BANK0.APEX_CONFIG0.dmp_mem_reset_en = 1

# Wait for DMP SRAM to be cleared
wait(1 ms)

# Set up OTP controller to reload factory-trimmed self-test response into SRAM
MREG1. ST_COPY_EN.st_copy_en = 3

# Take the OTP macro out of power-down mode
MREG2.OTP_CTRL7.otp_pwr_down = 0

# Wait for voltage generator to power on
Wait(100 µs)

# Trigger OTP to reload data (this time in self-test mode)
MREG2.OTP_CTRL7. OTP_RELOAD = 1

# Wait for OTP reload
Wait(20 µs)
```

```
# Set required self-test limit for accel and gyro
MREG1.ST_CONFIG.accel_st_lim = 7 # 50%
MREG1.ST_CONFIG.gyro_st_lim = 7 # 50%

# Set self-test number of samples
MREG1.ST_CONFIG.st_number_samples = 0 # 16 samples

# Write register to generate interrupt after both self-tests complete
BANK0.INT_SOURCE0.st_int1_en= 1

# Enable accel and/or gyro self-test.
# If both accel and gyro self-test are enabled, they should be set simultaneously in the same write access
MREG1.SELFTEST.accel_st_en = 1
MREG1.SELFTEST.gyro_st_en = 1

# Wait for st_done interrupt or poll int_status_st_done bit
while (BANK0.INT_STATUS.st_int == 0)

# Read self-test results
if (MREG1.ST_STATUS1.accel_st_pass == 1) ST is successful on accel
if (MREG1.ST_STATUS2.gyro_st_pass == 1) ST is successful on gyro

# Disable self-test
MREG1.SELFTEST.accel_st_en = 0
MREG1.SELFTEST.gyro_st_en = 0
```

4 EXAMPLE SELF-TEST SOFTWARE CODE

The below is an example self-test software code.

Please note, ST_STATUS1 and ST_STATUS2 must be read out consectively without new BLK_SEL_W and MADDRE_W config between the two M_R read. Bus address is internally buffered and auto-incremented for them.

```
void SelfTest_SPI()
{
uint8_t value[20],i;
value[0] = 0x10; spi_master_write_register(0x1f, 1, &value); delay_us(10);//gyro/accel=off, idle=1 (RCOSC)
delay_ms(10);
spi_master_read_register(0x00, 1, &value); delay_us(10);//check MCLK_RDY
while ((value[0] & 0x08) == 0){spi_master_read_register(0x00, 1, &value); delay_us(10);}
printf("MCLK ready %02x\r\n", value[0]);
value[0] = 0x01; spi_master_write_register(0x25, 1, &value); delay_us(10); //Clear DMP SRAM
delay_ms(1);

//OTP_CONFIG.otp_copy_mode = 3
value[0] = MREG_read(0x00, 0x2B); printf("ORG: OTP_CONFIG.otp_copy_mode %02x\r\n", value[0]);
value[1] = value[0] | 0x0C;
MREG_write(0x00, 0x2B, value[1]);
value[0] = MREG_read(0x00, 0x2B); printf("NEW: OTP_CONFIG.otp_copy_mode %02x\r\n\r\n", value[0]);

//OTP_CTRL7.otp_pwr_down = 0
value[0] = MREG_read(0x28, 0x06); printf("ORG: OTP_CTRL7.otp_pwr_down %02x\r\n", value[0]);
value[1] = value[0] & 0xFD;
MREG_write(0x28, 0x06, value[1]);
value[0] = MREG_read(0x28, 0x06); printf("NEW: OTP_CTRL7.otp_pwr_down (should be ORG & 0xFD) %02x\r\n\r\n", value[0]);

//Trigger OTP to reload data (this time in self-test mode), OTP_CTRL7.otp_reload = 1 (bit3)
value[0] = MREG_read(0x28, 0x06); printf("ORG: OTP_CTRL7.otp_reload %02x\r\n", value[0]);
value[1] = value[0] | 0x08;
MREG_write(0x28, 0x06, value[1]);
value[0] = MREG_read(0x28, 0x06); printf("NEW: OTP_CTRL7.otp_reload=1 (should be ORG|0x02) %02x\r\n\r\n", value[0]); // after
OPT load, OTP PWR is down.

//Set required self-test limit for accel and gyro
//ST_CONFIG.accel_st_lim = 7 (50%); ST_CONFIG.gyro_st_lim = 7 (50%)
//Set self-test number of samples. ST_CONFIG.st_num_samples = 0 (16 samples)
value[0] = 0x3F; MREG_write(0x00, 0x13, value[0]); //16 samples, 50% for G and A was 0x3f
value[0] = MREG_read(0x00, 0x13);printf("NEW: ST_CONFIG LIMIT (should be 0x3F) %02x\r\n\r\n", value[0]);

//Write register to generate interrupt after both self-tests complete
value[0] = 0x80; spi_master_write_register(0x2B, 1, &value); delay_us(10); //INT_SOURCE0.int_st_done_int1_en = 1

//Enable accel and/or gyro self-test. If both accel and gyro self-test are enabled, they should be set simultaneously in the same write
access
//SELFTEST.accel_st_en = 1; SELFTEST.gyro_st_en = 1
value[0] = 0xC0; MREG_write(0x00, 0x14, value[0]); //accel_st_en = 1, gyro_st_en = 1,
value[0] = MREG_read(0x00, 0x14);printf("NEW: SELFTEST EN (should be 0xC0) %02x\r\n", value[0]);

//Wait for st_done interrupt or poll int_status_st_done bit
spi_master_read_register(0x3A, 1, &value); delay_us(10); //check BANK0.INT_STATUS.st_int
while ((value[0] & 0x80) == 0){spi_master_read_register(0x3A, 1, &value);delay_us(10);}
printf("st_int done %02x\r\n", value[0]);

//Read self-test results
```

```
//if (ST_STATUS1.dmp_accel_st_pass == 1) ST is successful on accel
//if (ST_STATUS2.dmp_gyro_st_pass == 1) ST is successful on gyro
//Note, the ST_STATUS1 and ST_STATUS2 registers must be read out concectively to get correct result.
value[0] = 0x00; spi_master_write_register(0x7C, 1, &value); delay_us(10); //BLK_SEL_W=0
value[0] = 0x63; spi_master_write_register(0x7D, 1, &value); delay_us(10); //MADDR_R = 0x63, ST_STATUS1.dmp_accel_st_pass
spi_master_read_register(0x7E, 1, &value);      delay_us(10); //M_R, read ST_STATUS1
spi_master_read_register(0x7E, 1, &value[1]);  delay_us(10); //M_R,  consectively read ST_STATUS2
printf("ST_STATUS1/2.dmp_accel_st_pass and gyro_pass (bit5 should be 1) %02x, %02x\r\nn", value[0], value[1]);

//Disable self-test. MREG_TOP1.SELFTEST.accel_st_en = 0; MREG_TOP1.SELFTEST.gyro_st_en = 0
value[0] = 0x00; MREG_write(0x00, 0x14, value[0]); //accel_st_en = 1, gyro_st_en = 1,
value[0] = MREG_read(0x00, 0x14);      printf("Disable ST: SELFTEST (should be 0x00) %02x\r\n", value[0]);

}



uint8_t MREG_read(uint8_t BLK_SEL_R, uint8_t MADDR_R)
{
    uint8_t value[5];
    value[0] = BLK_SEL_R;   spi_master_write_register(0x7C, 1, &value[0]); delay_us(10);
    value[1] = MADDR_R;   spi_master_write_register(0x7D, 1, &value[1]); delay_us(10);
    spi_master_read_register(0x7E, 1, &value); delay_us(10);
value[2] = 0;               spi_master_write_register(0x7C, 1, &value[0]); delay_us(10); //restore default return value[0];
}

void MREG_write(uint8_t BLK_SEL_W, uint8_t MADDR_W, uint8_t data)
{
    uint8_t value[5];
    value[0] = BLK_SEL_W;   spi_master_write_register(0x79, 1, &value[0]); delay_us(10);
    value[1] = MADDR_W;   spi_master_write_register(0x7A, 1, &value[1]); delay_us(10);
    value[2] = data;            spi_master_write_register(0x7B, 1, &value[2]); delay_us(10);
    value[3] = 0x00;            spi_master_write_register (0x79, 1, &value); delay_us(10); //restore default
}
```

5 REVISION HISTORY

REVISION DATE	REVISION	DESCRIPTION
03/09/2021	1.0	Initial Release
03/23/2021	1.1	Added “ICM-42670x”

©2021 InvenSense. All rights reserved.