
LSM6DSV320X: finite state machine

Introduction

The **LSM6DSV320X** is a high-end, low-noise, low-power, small IMU, featuring a 3-axis digital low-*g* accelerometer at 16 *g*, a 3-axis digital high-*g* accelerometer at 320 *g*, and a 3-axis digital gyroscope, which offers the best IMU sensor with a quad-channel architecture for processing acceleration and angular rate data on four separate channels (user interface, OIS, EIS, and high-*g* accelerometer data) with dedicated configuration, processing, and filtering along with a dedicated high-*g* sensor for high-*g* shock and car crash detection.

This document provides information on the finite state machine feature available in the LSM6DSV320X. The finite state machine processing capability allows moving some algorithms from the application processor to the MEMS sensor, enabling consistent reduction of power consumption.

The processing capability of a finite state machine is achieved through logical connections. A state machine is a mathematical abstraction consisting of a finite number of states and transitions between them. The definition of the states and how they are connected implement the desired logic for pattern recognition.

The LSM6DSV320X can be configured to run up to eight finite state machines simultaneously and independently. The finite state machines are stored in the device and generate results in the dedicated output registers.

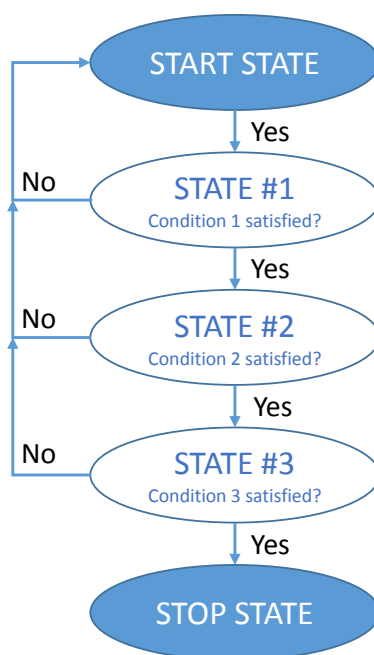
The results of the finite state machine can be read from the application processor at any time. Furthermore, there is the possibility to generate an interrupt when the desired pattern is detected.

1 Finite state machine (FSM)

1.1 Finite state machine definition

A finite state machine (FSM) is a mathematical abstraction used to design logic connections. It is a behavioral model composed of a finite number of states and transitions between states, similar to a flowchart in which it is possible to inspect the way logic runs when certain conditions are met. The state machine begins with a start state, goes to different states through transitions dependent on the inputs, and can finally end in a specific state (called stop state). The current state is determined by the past states of the system. The following figure depicts the flow of a generic state machine.

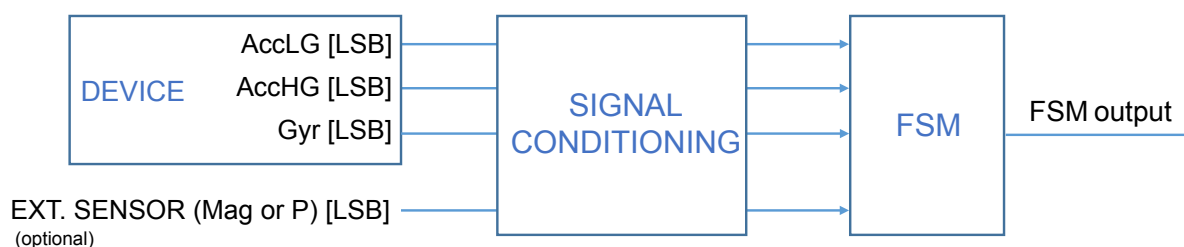
Figure 1. Generic state machine



1.2 Finite state machine in the LSM6DSV320X

The LSM6DSV320X is a small IMU featuring a 3-axis digital low-*g* accelerometer at 16 *g*, a 3-axis digital high-*g* accelerometer at 320 *g*, and a 3-axis digital gyroscope. It is also possible to connect an external sensor like a magnetometer or pressure sensor by using the sensor hub feature (mode 2). These data can be used as input of up to eight programs in the embedded finite state machine (refer to the following figure).

Figure 2. State machine in the LSM6DSV320X



The FSM structure is highly modular: it is possible to easily write up to eight programs, each one able to recognize a specific gesture.

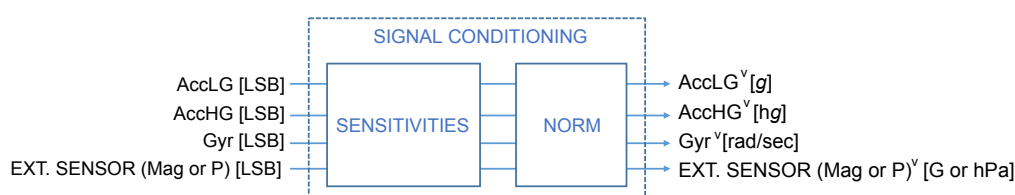
All eight finite state machines are independent: each one has its dedicated memory area and it is independently executed. An interrupt is generated when the end state is reached or when some specific command is performed. Typically, the interrupt is generated when a specific gesture is recognized.

2 Signal conditioning block

The signal conditioning block is shown in the following figure and it is used as the interface between incoming sensor data and the FSM block. This block is needed to convert the output sensor data (represented in [LSB]) with the following unit conventions:

- Low-*g* accelerometer data in [*g*]
- High-*g* accelerometer data in [*hg*]
- Gyroscope data in [rad/sec]
- External sensor: if it is a magnetometer, data have to be converted to [*G*]. If it is a pressure sensor, data have to be converted to [*hPa*].

Figure 3. Signal conditioning block



This block is intended to apply the sensitivity to [LSB] input data, and then convert these data in half-precision floating-point (HFP) format before passing them to the FSM block. In greater detail:

- Low-*g* accelerometer data conversion factor of the LSM6DSV320X is automatically handled by the device.
- High-*g* accelerometer data conversion factor of the LSM6DSV320X is automatically handled by the device if the selected full-scale range is equal to or lower than $\pm 256\text{ g}$; otherwise, if the selected full-scale range is equal to $\pm 320\text{ g}$, the host must write in the XL_HG_SENSITIVITY_L (58h) and XL_HG_SENSITIVITY_H (59h) embedded advanced features registers the value 0x06D4.
- Gyroscope data conversion factor of the LSM6DSV320X is automatically handled by the device.

- External sensor data conversion factor is not automatically handled by the device. The LSM6DSV320X FSM supports processing external sensor data in two different formats as listed below:
 - 3-axis sensors with 16-bit data like magnetometer sensors, which are enabled by setting the EXT_FORMAT_SEL bit of the EXT_FORMAT (00h) embedded advanced features register to 0 (default value). In this case, the external sensor data are internally processed as follows.
 1. Raw external sensor data are converted in HFP format (no sensitivity is applied).
 2. Hard-iron offset is applied. The FSM_EXT_OFFX_L (C0h), FSM_EXT_OFFX_H (C1h), FSM_EXT_OFFY_L (C2h), FSM_EXT_OFFY_H (C3h), FSM_EXT_OFFZ_L (C4h), and FSM_EXT_OFFZ_H (C5h) embedded advanced features registers must contain the raw hard-iron offset in HFP format.
 3. Soft-iron compensation is applied. The FSM_EXT_MATRIX_XX_L (C6h), FSM_EXT_MATRIX_XX_H (C7h), FSM_EXT_MATRIX_XY_L (C8h), FSM_EXT_MATRIX_XY_H (C9h), FSM_EXT_MATRIX_XZ_L (CAh), FSM_EXT_MATRIX_XZ_H (CBh), FSM_EXT_MATRIX_YY_L (CCh), FSM_EXT_MATRIX_YY_H (CDh), FSM_EXT_MATRIX_YZ_L (CEh), FSM_EXT_MATRIX_YZ_H (CFh), FSM_EXT_MATRIX_ZZ_L (D0h), and FSM_EXT_MATRIX_ZZ_H (D1h) embedded advanced features registers must contain the soft-iron coefficients in HFP format.
 4. Sensitivity is applied. The FSM_EXT_SENSITIVITY_L (BAh) and FSM_EXT_SENSITIVITY_H (BBh) embedded advanced features registers must contain the sensitivity value in HFP format.
 5. Data are rotated. The EXT_CFG_A (D4h) and EXT_CFG_B (D5h) embedded advanced features registers must contain the external sensor orientation configuration.
 - 1-axis sensors with 24-bit data like pressure sensors, which are enabled by setting the EXT_FORMAT_SEL bit of the EXT_FORMAT (00h) embedded advanced features register to 1. In this case, the external sensor data are internally processed as follows.
 1. Raw external sensor data is kept in LSB format (no sensitivity is applied).
 2. Offset is applied. The EXT_3BYTE_OFFSET_XL (06h), EXT_3BYTE_OFFSET_L (07h), and EXT_3BYTE_OFFSET_H (08h) embedded advanced features registers must contain the offset in LSB format.
 3. Sensitivity is applied. The EXT_3BYTE_SENSITIVITY_L (02h) and EXT_3BYTE_SENSITIVITY_H (03h) embedded advanced features registers must contain the sensitivity value in HFP format.

Note: *Magnetometer data have to be converted to [G], while pressure data have to be converted to [hPa].*

Example: LIS2MDL magnetometer sensitivity is 1.5 mG/LSB → 0.0015 G/LSB → 1624h HFP; this is the default external sensor sensitivity value for the LSM6DSV320X.

Procedure to change the conversion factor for an external sensor:

- | | |
|--|--|
| 1. Write 80h to register 01h | // Enable embedded functions registers access |
| 2. Write 40h to register 17h | // PAGE_RW (17h) = 40h: enable write operation |
| 3. Write 01h to register 02h | // PAGE_SEL (02h) = 01h: select embedded advanced features registers page 0 |
| 4. Write BAh to register 08h | // PAGE_ADDRESS (08h) = BAh (FSM_EXT_SENSITIVITY_L address) |
| 5. Write [LSB] conversion factor to register 09h | // Write [LSB] conversion factor value to register FSM_EXT_SENSITIVITY_L (BAh) |
| 6. Write [MSB] conversion factor to register 09h | // Write [MSB] conversion factor value to register FSM_EXT_SENSITIVITY_H (BBh) |
| 7. Write 01h to register 02h | // PAGE_SEL (02h) = 01h: select embedded advanced features registers page 0 |
| 8. Write 00h to register 17h | // PAGE_RW (17h) = 00h: disable read / write operation |
| 9. Write 00h to register 01h | // Disable embedded functions registers access |

In addition to the conversion to HFP format, the signal conditioning block computes the norm of the input data, defined as follows:

$$V = \sqrt{x^2 + y^2 + z^2}$$

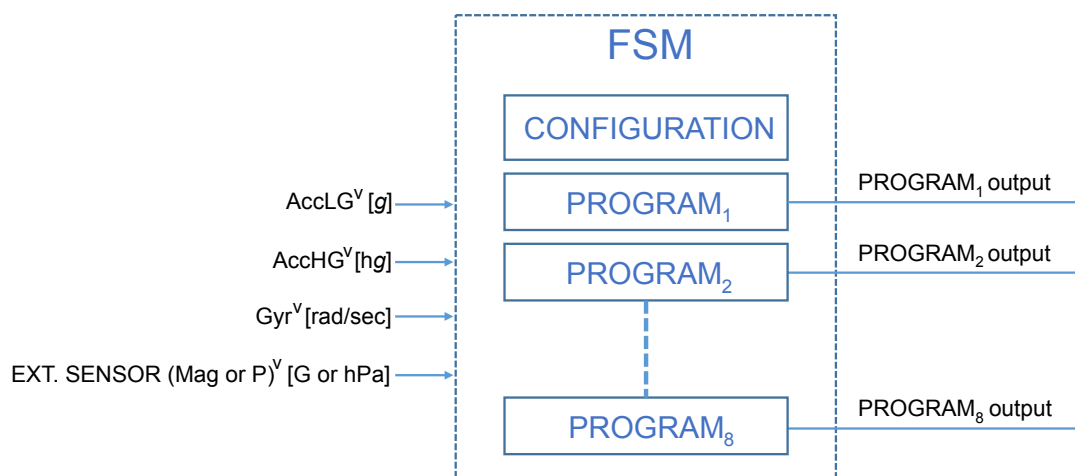
The norm of the input data can be used in the state machine programs, in order to guarantee a high level of program customization for the user.

3 FSM block

Output data signals coming from the signal conditioning block are sent to the FSM block that is detailed in the following figure. The FSM block is mainly composed of:

- A general FSM configuration block: it affects all programs and includes some registers that have to be properly initialized in order to configure and customize the entire FSM block.
- A maximum of eight configurable programs: each program processes input data and generates an output.

Figure 4. FSM block



FSM configuration and program blocks are described in the following sections.

3.1 Configuration block

The configuration block is composed of a set of registers involved in the FSM configuration (FSM ODR, interrupts, programs configuration, and so forth).

The embedded functions registers can be used to properly configure the FSM: these registers are accessible when the FUNC_CFG_EN bit is set to 1 and the SHUB_REG_ACCESS bit is set to 0 in the FUNC_CFG_ACCESS (01h) register.

The LSM6DSV320X device is provided with an extended number of registers inside the embedded functions register set, called embedded advanced features registers, that are divided in pages. A specific read / write procedure must be followed to access the embedded features registers. Registers involved in this specific procedure are the following:

- PAGE_SEL (02h): it selects the desired page.
- PAGE_ADDRESS (08h): it selects the desired register address in the selected page.
- PAGE_VALUE (09h): it sets the value to be written in the selected register (only in write operation).
- PAGE_RW (17h): it is used to select the read / write operation.

The script below shows the generic procedure to write a YYh value in the register having address XXh inside the page number Z of the embedded features registers set:

1. Write 80h to register 01h // Enable embedded functions registers access
2. Write 40h to register 17h // PAGE_RW (17h) = 40h: enable write operation
3. Write Z1h to register 02h // PAGE_SEL (02h) = Z1h: select embedded advanced features registers page Z
4. Write XXh to register 08h // PAGE_ADDRESS (08h) = XXh: XXh is the address of the register to be configured
5. Write YYh to register 09h // PAGE_VALUE (09h) = YYh: YYh is the value to be written
6. Write 01h to register 02h // PAGE_SEL (02h) = 01h: select embedded advanced features registers page 0. This is needed for the correct operation of the device.
7. Write 00h to register 17h // PAGE_RW (17h) = 00h: disable read / write operation
8. Write 00h to register 01h // Disable embedded functions registers access

Note: *After a write transaction, the PAGE_ADDRESS (08h) register is automatically incremented.*

Program configurations must be written in the embedded advanced features registers, starting from the register address indicated by the FSM_START_ADD_L (7Eh) and FSM_START_ADD_H (7Fh) registers. All programs have to be written in consecutive registers, including two important aspects:

- Both the PAGE_SEL (02h) register and PAGE_ADDRESS (08h) register have to be properly updated when moving from one page to another (that is, when passing from page 03h, address FFh to page 04h, address 00h). The LSM6DSV320X device provides eight pages that can be addressed through the PAGE_SEL (02h) register. To address the last page, PAGE_SEL (02h) has to be set to 71h.
- Program SIZE byte must be an even number. If it is odd, an additional STOP state has to be added at the end of the instruction section.

Once the program configurations have been written, the FSM programs are executed at the rate configured through the FSM_ODR_[2:0] bits of the FSM_ODR (5Fh) embedded functions register, as explained below:

- If the configured FSM execution rate is equal to the fastest active sensor ODR, the FSM programs are executed at the rate of the fastest active sensor ODR, which matches the FSM execution rate.
- If the configured FSM execution rate is lower than the fastest active sensor ODR, the FSM programs are executed at the rate of the FSM execution rate. In this case, the FSM decimates the input data.
- If the configured FSM execution rate is greater than the fastest active sensor ODR, the FSM programs are executed at the rate of the fastest active sensor ODR.
- If all the sensors are configured in power-down mode (there is no active sensor), the FSM programs are not executed.

For a detailed example of how to configure the entire FSM, refer to [Section 11: FSM configuration example](#).

3.1.1

Registers

All the FSM-related registers given in the following table are accessible from the primary SPI/I²C/MIPI I3C[®] interface only.

Table 1. Registers

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FUNC_CFG_ACCESS	01h	EMB_FUNC_REG_ACCESS	SHUB_REG_ACCESS	0	0	FSM_WR_CTRL_EN	-	-	-
CTRL_STATUS	1Ah	0	0	0	0	0	FSM_WR_CTRL_STATUS	-	0
EMB_FUNC_STATUS_MAINPAGE	49h	IS_FSM_LC	0	-	-	-	0	0	0
FSM_STATUS_MAINPAGE	4Ah	IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1
MD1_CFG	5Eh	-	-	-	-	-	-	INT1_EMB_FUNC	-
MD2_CFG	5Fh	-	-	-	-	-	-	INT2_EMB_FUNC	-



3.1.2

Embedded functions registers

The table given below provides a list of the FSM-related registers for the embedded functions available in the device and the corresponding addresses. Embedded functions registers are accessible when the EMB_FUNC_REG_ACCESS bit is set to 1 in the FUNC_CFG_ACCESS (01h) register.

Table 2. Embedded functions registers

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PAGE_SEL	02h	PAGE_SEL3	PAGE_SEL2	PAGE_SEL1	PAGE_SEL0	0	0	0	1
EMB_FUNC_EN_B	05h	0	0	0	-	-	0	0	FSM_EN
PAGE_ADDRESS	08h	PAGE_ADDR7	PAGE_ADDR6	PAGE_ADDR5	PAGE_ADDR4	PAGE_ADDR3	PAGE_ADDR2	PAGE_ADDR1	PAGE_ADDR0
PAGE_VALUE	09h	PAGE_VALUE7	PAGE_VALUE6	PAGE_VALUE5	PAGE_VALUE4	PAGE_VALUE3	PAGE_VALUE2	PAGE_VALUE1	PAGE_VALUE0
EMB_FUNC_INT1	0Ah	INT1_FSM_LC	0	-	-	-	0	0	0
FSM_INT1	0Bh	INT1_FSM8	INT1_FSM7	INT1_FSM6	INT1_FSM5	INT1_FSM4	INT1_FSM3	INT1_FSM2	INT1_FSM1
EMB_FUNC_INT2	0Eh	INT2_FSM_LC	0	-	-	-	0	0	0
FSM_INT2	0Fh	INT2_FSM8	INT2_FSM7	INT2_FSM6	INT2_FSM5	INT2_FSM4	INT2_FSM3	INT2_FSM2	INT2_FSM1
EMB_FUNC_STATUS	12h	IS_FSM_LC	0	-	-	-	0	0	0
FSM_STATUS	13h	IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1
PAGE_RW	17h	EMB_FUNC_LIR	PAGE_WRITE	PAGE_READ	0	0	0	0	0
EMB_FUNC_FIFO_EN_B	45h	0	0	0	0	0	FSM_FIFO_EN	-	0
FSM_ENABLE	46h	FSM8_EN	FSM7_EN	FSM6_EN	FSM5_EN	FSM4_EN	FSM3_EN	FSM2_EN	FSM1_EN
FSM_LONG_COUNTER_L	48h	FSM_LC_7	FSM_LC_6	FSM_LC_5	FSM_LC_4	FSM_LC_3	FSM_LC_2	FSM_LC_1	FSM_LC_0
FSM_LONG_COUNTER_H	49h	FSM_LC_15	FSM_LC_14	FSM_LC_13	FSM_LC_12	FSM_LC_11	FSM_LC_10	FSM_LC_9	FSM_LC_8
FSM_OUTS1	4Ch	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS2	4Dh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS3	4Eh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS4	4Fh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS5	50h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS6	51h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS7	52h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS8	53h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_ODR	5Fh	0	1	FSM_ODR_2	FSM_ODR_1	FSM_ODR_0	0	1	1
EMB_FUNC_INIT_B	67h	0	0	0	-	-	PT_INIT	0	FSM_INIT



3.1.3

Embedded advanced features pages

The table given below provides a list of the FSM-related registers for the embedded advanced features page 0. These registers are accessible when PAGE_SEL[3:0] are set to 0000 in the PAGE_SEL (02h) register.

Table 3. Embedded advanced features registers - page 0

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_EXT_SENSITIVITY_L	BAh	FSM_EXT_S_7	FSM_EXT_S_6	FSM_EXT_S_5	FSM_EXT_S_4	FSM_EXT_S_3	FSM_EXT_S_2	FSM_EXT_S_1	FSM_EXT_S_0
FSM_EXT_SENSITIVITY_H	BBh	FSM_EXT_S_15	FSM_EXT_S_14	FSM_EXT_S_13	FSM_EXT_S_12	FSM_EXT_S_11	FSM_EXT_S_10	FSM_EXT_S_9	FSM_EXT_S_8
FSM_EXT_OFFX_L	C0h	FSM_EXT_OFFX_7	FSM_EXT_OFFX_6	FSM_EXT_OFFX_5	FSM_EXT_OFFX_4	FSM_EXT_OFFX_3	FSM_EXT_OFFX_2	FSM_EXT_OFFX_1	FSM_EXT_OFFX_0
FSM_EXT_OFFX_H	C1h	FSM_EXT_OFFX_15	FSM_EXT_OFFX_14	FSM_EXT_OFFX_13	FSM_EXT_OFFX_12	FSM_EXT_OFFX_11	FSM_EXT_OFFX_10	FSM_EXT_OFFX_9	FSM_EXT_OFFX_8
FSM_EXT_OFFY_L	C2h	FSM_EXT_OFFY_7	FSM_EXT_OFFY_6	FSM_EXT_OFFY_5	FSM_EXT_OFFY_4	FSM_EXT_OFFY_3	FSM_EXT_OFFY_2	FSM_EXT_OFFY_1	FSM_EXT_OFFY_0
FSM_EXT_OFFY_H	C3h	FSM_EXT_OFFY_15	FSM_EXT_OFFY_14	FSM_EXT_OFFY_13	FSM_EXT_OFFY_12	FSM_EXT_OFFY_11	FSM_EXT_OFFY_10	FSM_EXT_OFFY_9	FSM_EXT_OFFY_8
FSM_EXT_OFFZ_L	C4h	FSM_EXT_OFFZ_7	FSM_EXT_OFFZ_6	FSM_EXT_OFFZ_5	FSM_EXT_OFFZ_4	FSM_EXT_OFFZ_3	FSM_EXT_OFFZ_2	FSM_EXT_OFFZ_1	FSM_EXT_OFFZ_0
FSM_EXT_OFFZ_H	C5h	FSM_EXT_OFFZ_15	FSM_EXT_OFFZ_14	FSM_EXT_OFFZ_13	FSM_EXT_OFFZ_12	FSM_EXT_OFFZ_11	FSM_EXT_OFFZ_10	FSM_EXT_OFFZ_9	FSM_EXT_OFFZ_8
FSM_EXT_MATRIX_XX_L	C6h	FSM_EXT_MAT_XX_7	FSM_EXT_MAT_XX_6	FSM_EXT_MAT_XX_5	FSM_EXT_MAT_XX_4	FSM_EXT_MAT_XX_3	FSM_EXT_MAT_XX_2	FSM_EXT_MAT_XX_1	FSM_EXT_MAT_XX_0
FSM_EXT_MATRIX_XX_H	C7h	FSM_EXT_MAT_XX_15	FSM_EXT_MAT_XX_14	FSM_EXT_MAT_XX_13	FSM_EXT_MAT_XX_12	FSM_EXT_MAT_XX_11	FSM_EXT_MAT_XX_10	FSM_EXT_MAT_XX_9	FSM_EXT_MAT_XX_8
FSM_EXT_MATRIX_XY_L	C8h	FSM_EXT_MAT_XY_7	FSM_EXT_MAT_XY_6	FSM_EXT_MAT_XY_5	FSM_EXT_MAT_XY_4	FSM_EXT_MAT_XY_3	FSM_EXT_MAT_XY_2	FSM_EXT_MAT_XY_1	FSM_EXT_MAT_XY_0
FSM_EXT_MATRIX_XY_H	C9h	FSM_EXT_MAT_XY_15	FSM_EXT_MAT_XY_14	FSM_EXT_MAT_XY_13	FSM_EXT_MAT_XY_12	FSM_EXT_MAT_XY_11	FSM_EXT_MAT_XY_10	FSM_EXT_MAT_XY_9	FSM_EXT_MAT_XY_8
FSM_EXT_MATRIX_XZ_L	CAh	FSM_EXT_MAT_XZ_7	FSM_EXT_MAT_XZ_6	FSM_EXT_MAT_XZ_5	FSM_EXT_MAT_XZ_4	FSM_EXT_MAT_XZ_3	FSM_EXT_MAG_SI_XZ_2	FSM_EXT_MAT_XZ_1	FSM_EXT_MAT_XZ_0
FSM_EXT_MATRIX_XZ_H	CBh	FSM_EXT_MAT_XZ_15	FSM_EXT_MAT_XZ_14	FSM_EXT_MAT_XZ_13	FSM_EXT_MAT_XZ_12	FSM_EXT_MAT_XZ_11	FSM_EXT_MAT_XZ_10	FSM_EXT_MAT_XZ_9	FSM_EXT_MAT_XZ_8
FSM_EXT_MATRIX_YY_L	CCh	FSM_EXT_MAT_YY_7	FSM_EXT_MAT_YY_6	FSM_EXT_MAT_YY_5	FSM_EXT_MAT_YY_4	FSM_EXT_MAT_YY_3	FSM_EXT_MAT_YY_2	FSM_EXT_MAT_YY_1	FSM_EXT_MAT_YY_0
FSM_EXT_MATRIX_YY_H	CDh	FSM_EXT_MAT_YY_15	FSM_EXT_MAT_YY_14	FSM_EXT_MAT_YY_13	FSM_EXT_MAT_YY_12	FSM_EXT_MAT_YY_11	FSM_EXT_MAT_YY_10	FSM_EXT_MAT_YY_9	FSM_EXT_MAT_YY_8
FSM_EXT_MATRIX_YZ_L	CEh	FSM_EXT_MAT_YZ_7	FSM_EXT_MAT_YZ_6	FSM_EXT_MAT_YZ_5	FSM_EXT_MAT_YZ_4	FSM_EXT_MAT_YZ_3	FSM_EXT_MAT_YZ_2	FSM_EXT_MAT_YZ_1	FSM_EXT_MAT_YZ_0
FSM_EXT_MATRIX_YZ_H	CFh	FSM_EXT_MAT_YZ_15	FSM_EXT_MAT_YZ_14	FSM_EXT_MAT_YZ_13	FSM_EXT_MAT_YZ_12	FSM_EXT_MAT_YZ_11	FSM_EXT_MAT_YZ_10	FSM_EXT_MAT_YZ_9	FSM_EXT_MAT_YZ_8
FSM_EXT_MATRIX_ZZ_L	D0h	FSM_EXT_MAT_ZZ_7	FSM_EXT_MAT_ZZ_6	FSM_EXT_MAT_ZZ_5	FSM_EXT_MAT_ZZ_4	FSM_EXT_MAT_ZZ_3	FSM_EXT_MAT_ZZ_2	FSM_EXT_MAT_ZZ_1	FSM_EXT_MAT_ZZ_0
FSM_EXT_MATRIX_ZZ_H	D1h	FSM_EXT_MAT_ZZ_15	FSM_EXT_MAT_ZZ_14	FSM_EXT_MAT_ZZ_13	FSM_EXT_MAT_ZZ_12	FSM_EXT_MAT_ZZ_11	FSM_EXT_MAT_ZZ_10	FSM_EXT_MAT_ZZ_9	FSM_EXT_MAT_ZZ_8
EXT_CFG_A	D4h	0	EXT_Y_AXIS2	EXT_Y_AXIS1	EXT_Y_AXIS0	0	EXT_Z_AXIS2	EXT_Z_AXIS1	EXT_Z_AXIS0
EXT_CFG_B	D5h	0	0	0	0	0	EXT_X_AXIS2	EXT_X_AXIS1	EXT_X_AXIS0



The table given below provides a list of the FSM-related registers for the embedded advanced features page 1. These registers are accessible when PAGE_SEL[3:0] are set to 0001 in the PAGE_SEL (02h) register.

Table 4. Embedded advanced features registers - page 1

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
XL_HG_SENSITIVITY_L	58h	XL_HG_S_7	XL_HG_S_6	XL_HG_S_5	XL_HG_S_4	XL_HG_S_3	XL_HG_S_2	XL_HG_S_1	XL_HG_S_0
XL_HG_SENSITIVITY_H	59h	XL_HG_S_15	XL_HG_S_14	XL_HG_S_13	XL_HG_S_12	XL_HG_S_11	XL_HG_S_10	XL_HG_S_9	XL_HG_S_8
FSM_LC_TIMEOUT_L	7Ah	FSM_LC_TIMEOUT7	FSM_LC_TIMEOUT6	FSM_LC_TIMEOUT5	FSM_LC_TIMEOUT4	FSM_LC_TIMEOUT3	FSM_LC_TIMEOUT2	FSM_LC_TIMEOUT1	FSM_LC_TIMEOUT0
FSM_LC_TIMEOUT_H	7Bh	FSM_LC_TIMEOUT15	FSM_LC_TIMEOUT14	FSM_LC_TIMEOUT13	FSM_LC_TIMEOUT12	FSM_LC_TIMEOUT11	FSM_LC_TIMEOUT10	FSM_LC_TIMEOUT9	FSM_LC_TIMEOUT8
FSM_PROGRAMS	7Ch	FSM_N_PROG7	FSM_N_PROG6	FSM_N_PROG5	FSM_N_PROG4	FSM_N_PROG3	FSM_N_PROG2	FSM_N_PROG1	FSM_N_PROG0
FSM_START_ADD_L	7Eh	FSM_START7	FSM_START6	FSM_START5	FSM_START4	FSM_START3	FSM_START2	FSM_START1	FSM_START0
FSM_START_ADD_H	7Fh	FSM_START15	FSM_START14	FSM_START13	FSM_START12	FSM_START11	FSM_START10	FSM_START9	FSM_START8

The table given below provides a list of the FSM-related registers for the embedded advanced features page 2. These registers are accessible when PAGE_SEL[3:0] are set to 0010 in the PAGE_SEL (02h) register.

Table 5. Embedded advanced features registers - page 2

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EXT_FORMAT	00h	0	0	0	0	0	0	EXT_FORMAT_SEL	0
EXT_3BYTE_SENSITIVITY_L	02h	EXT_3BYTE_S_7	EXT_3BYTE_S_6	EXT_3BYTE_S_5	EXT_3BYTE_S_4	EXT_3BYTE_S_3	EXT_3BYTE_S_2	EXT_3BYTE_S_1	EXT_3BYTE_S_0
EXT_3BYTE_SENSITIVITY_H	03h	EXT_3BYTE_S_15	EXT_3BYTE_S_14	EXT_3BYTE_S_13	EXT_3BYTE_S_12	EXT_3BYTE_S_11	EXT_3BYTE_S_10	EXT_3BYTE_S_9	EXT_3BYTE_S_8
EXT_3BYTE_OFFSET_XL	06h	EXT_3BYTE_OFF_7	EXT_3BYTE_OFF_6	EXT_3BYTE_OFF_5	EXT_3BYTE_OFF_4	EXT_3BYTE_OFF_3	EXT_3BYTE_OFF_2	EXT_3BYTE_OFF_1	EXT_3BYTE_OFF_0
EXT_3BYTE_OFFSET_L	07h	EXT_3BYTE_OFF_15	EXT_3BYTE_OFF_14	EXT_3BYTE_OFF_13	EXT_3BYTE_OFF_12	EXT_3BYTE_OFF_11	EXT_3BYTE_OFF_10	EXT_3BYTE_OFF_9	EXT_3BYTE_OFF_8
EXT_3BYTE_OFFSET_H	08h	EXT_3BYTE_OFF_23	EXT_3BYTE_OFF_22	EXT_3BYTE_OFF_21	EXT_3BYTE_OFF_20	EXT_3BYTE_OFF_19	EXT_3BYTE_OFF_18	EXT_3BYTE_OFF_17	EXT_3BYTE_OFF_16

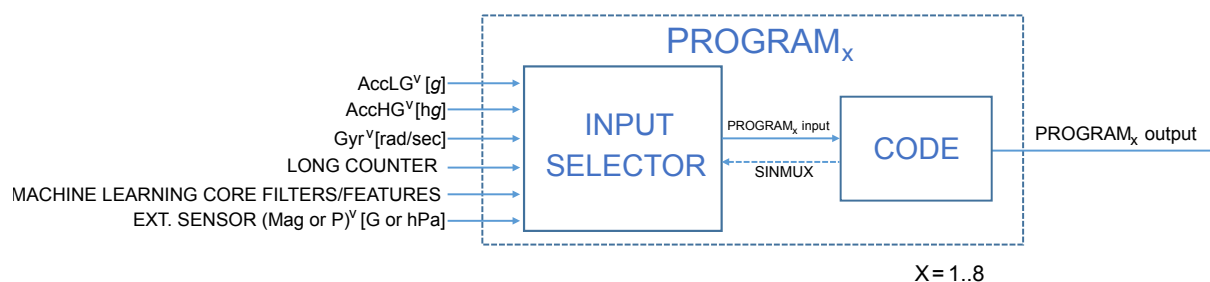


3.2 Program block

Output data coming from the signal conditioning block are sent to the FSM block, composed of eight program blocks. Each program block, as shown in the following figure, consists of:

- An input selector block, which selects the desired input data signal that is processed by the program
- A code block, composed of the data and the instructions that are executed

Figure 5. Program block



3.2.1 Input selector block

The input selector block allows the selection of the input data signal between the following physical sensor data signals or internally calculated data signals:

- LSM6DSV320X low-*g* accelerometer data, with precomputed norm (V)
- LSM6DSV320X high-*g* accelerometer data, with precomputed norm (V)
- LSM6DSV320X gyroscope data, with precomputed norm (V)
- External 3-axis sensors with 16-bit data like magnetometer sensors, with precomputed norm (V)
- External 1-axis sensors with 24-bit data like pressure sensors
- Long counter value
- Internally filtered data and computed features, by properly configuring the machine learning core
- Internally calculated angles, with precomputed norm (V)

The norm (V) is internally computed with the following formula:

$$V = \sqrt{x^2 + y^2 + z^2}$$

The machine learning core allows configuring the device to compute features (like average, variance, peak-to-peak, energy, and so forth) or filters (like high pass, band pass, IIR1 and IIR2) applied to internal / external sensor data. For more details about the machine learning core capabilities, refer to application note AN6292.

The following figures show the inputs of the finite state machine block in the low-*g* accelerometer, high-*g* accelerometer, and gyroscope digital chains. The position of the finite state machine (FSM) block in the digital chains is the same for all four connection modes available in the LSM6DSV320X.

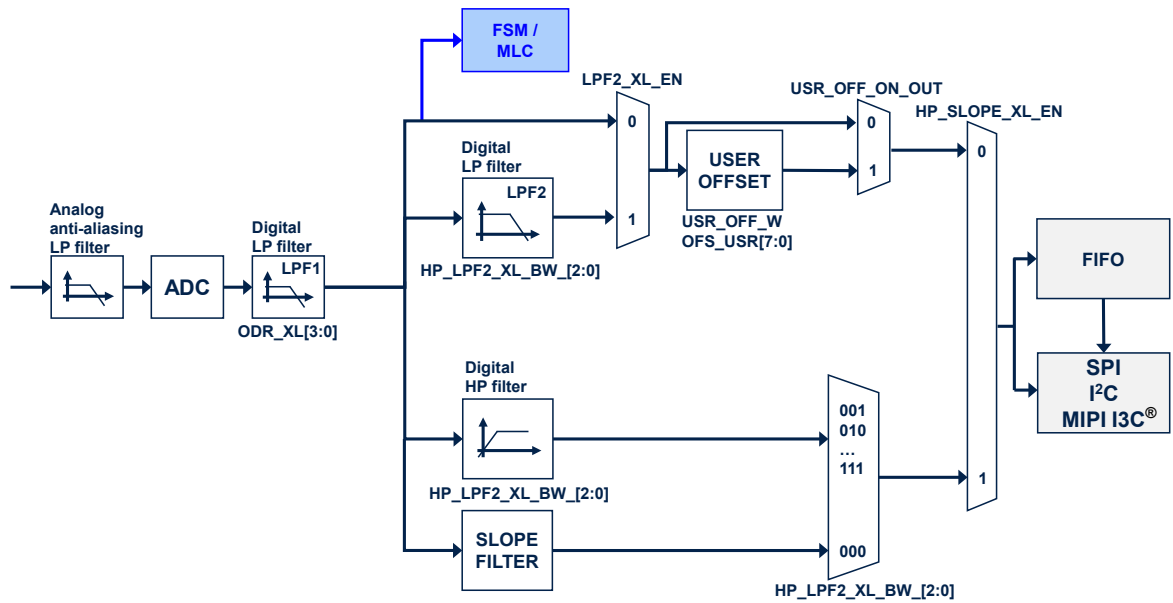
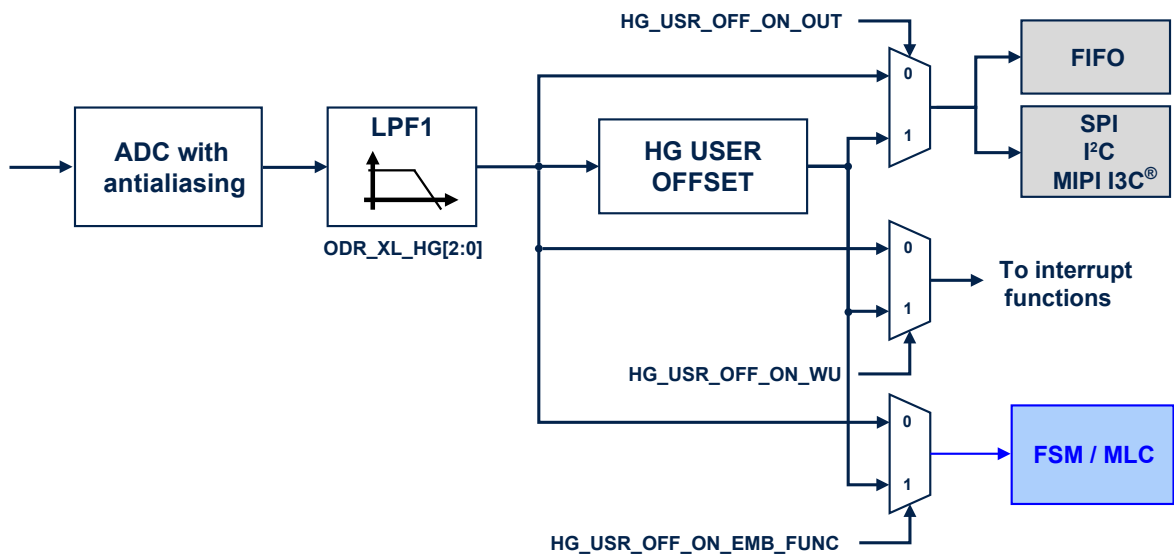
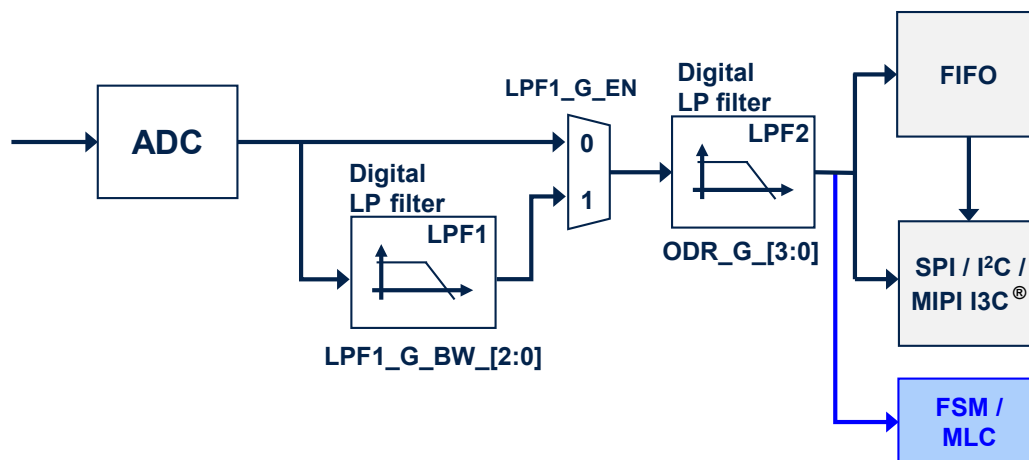
Figure 6. FSM inputs (low-g accelerometer)

Figure 7. FSM inputs (high-g accelerometer)


Figure 8. FSM inputs (gyroscope)


The signal bandwidth of the accelerometer and gyroscope depends on the device configuration. For additional information, refer to AN6119 available at www.st.com. The program block executes the configured program (code block) by processing the selected input signal and generating the corresponding program output signals, according to the purpose of the program.

Note: *The SINMUX command can be used by the user inside the program instructions section to dynamically switch the desired input signal for the program block. Refer to [SINMUX \(23h\)](#) for additional and detailed information about the SINMUX command.*

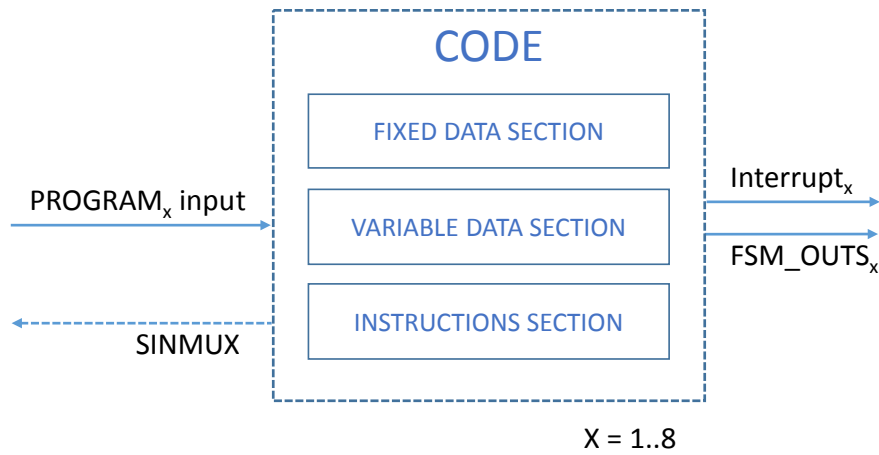
3.2.2 Code block

The FSM program_x code block contains the state machine program. The structure of a single program is shown in the following figure. It is composed of:

- A data section, composed of a fixed part (same size for all the FSMs) and a variable part (specific size for each FSM)
- An instructions section, composed of conditions and commands

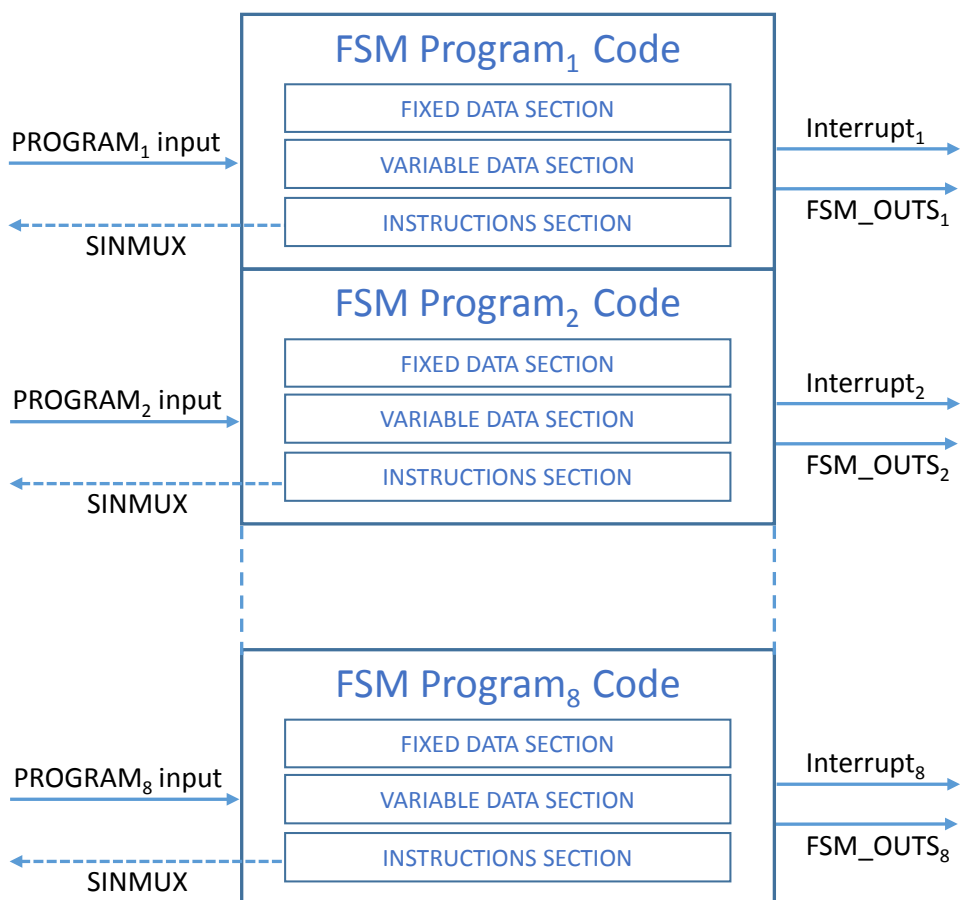
Each program can generate an interrupt_x signal and modify the corresponding FSM_OUTS_x register value, according to processed sample sets coming from the PROGRAM_x input signal that can be selected by issuing a specific SINMUX command.

Figure 9. FSM program_x code structure



All FSM programs are stored consecutively in a set of reserved embedded advanced features registers, as shown in the following figure. The maximum allowed size for each program is 256 bytes.

Note: *FSMs (according to all the embedded functions) have to be reconfigured each time the device is powered on.*

Figure 10. FSM program_x memory area


4 FSM interrupt status and signal

The FSM supports generating two different interrupt signals: the FSM program interrupt signal and the FSM long counter interrupt signal.

The FSM program interrupt signal is generated when the end state is reached (STOP command) or when some specific command is performed (OUTC / CONT / CONTREL commands). When an FSM program interrupt is generated, the corresponding temporary mask value is transmitted to its corresponding FSM_OUTS embedded functions register.

The FSM long counter interrupt signal is generated when the long counter value, stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) embedded functions registers, reaches the configured long counter timeout value stored in the FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh) embedded advanced features registers.

The FSM interrupt status can be checked by reading the dedicated register:

- FSM_STATUS_MAINPAGE (4Ah) register or FSM_STATUS (13h) embedded functions register for the FSM interrupt status
- EMB_FUNC_STATUS_MAINPAGE (49h) register or EMB_FUNC_STATUS (12h) embedded functions register for the long counter interrupt status

The FSM interrupt signal can be driven to the INT1/INT2 interrupt pin by setting the dedicated bit:

- INT1_FSM[1:8]/INT2_FSM[1:8] bit of the FSM_INT1/FSM_INT2 embedded functions register to 1
- INT1_FSM_LC/INT2_FSM_LC bit of the EMB_FUNC_INT1/EMB_FUNC_INT2 embedded functions register to 1

Note:

In both of the above cases it is mandatory to also enable routing the embedded functions events to the INT1/INT2 interrupt pin by setting the INT1_EMB_FUNC/INT2_EMB_FUNC bit of the MD1_CFG/MD2_CFG register.

The behavior of the interrupt signal is pulsed by default. The duration of the pulse is equal to $1 / \text{MAX_RATE}$ seconds, where MAX_RATE denotes the maximum rate of the enabled embedded functions.

Latched mode can be enabled by setting the EMB_FUNC_LIR bit of the PAGE_RW (17h) embedded functions register to 1. In this case, the interrupt signal and the status bit are reset when reading:

- FSM_STATUS_MAINPAGE (4Ah) register or FSM_STATUS (13h) embedded functions register for the FSM interrupt
- EMB_FUNC_STATUS_MAINPAGE (49h) register or EMB_FUNC_STATUS (12h) embedded functions register for the long counter interrupt

5 Long counter

The long counter is a 15-bit temporary counter resource shared by all the available FSM programs and available to the user. Each FSM program can increment, decrement, or reset the long counter value by using the INCR, DECR, or RESET command, respectively. The long counter value is stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) embedded functions registers. The minimum (and default) long counter value is 0, while the maximum long counter value is the configured timeout value stored in the FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh) embedded advanced features registers.

When the long counter value is equal to the configured long counter timeout value, the IS_FSM_LC status bit of the EMB_FUNC_STATUS_MAINPAGE (49h) register and EMB_FUNC_STATUS (12h) embedded functions register is set to 1.

Details about the FSM long counter interrupt are available in [Section 4: FSM interrupt status and signal](#).

This resource is common to all programs and does not need additional allocated resources in the **[Variable Data Section]**.

Note: When FSM_LC_TIMEOUT is equal to 0, the long counter feature is disabled.

Note: The FSM_LC_TIMEOUT value must be set lower than 2^{15} in 15-bit unsigned format.

6 Peak tracking

The LSM6DSV320X device allows detecting the intensity of shock events by exploiting the new peak-tracking feature, which is particularly useful in car crash detection or fall detection applications.

The high-*g* accelerometer peak-tracking feature allows monitoring the magnitude of the high-*g* accelerometer within a time window that can be defined by issuing dedicated commands from the finite state machine. When issuing the command related to the end of the monitoring window, this feature stores the high-*g* accelerometer three-axis components of the detected peak in the FIFO with a dedicated TAG.

[Section 7.2: Peak-tracking outputs in FIFO](#) and [Section 10.2.8: SETM \(F9h\)](#) show more details on its usage: an example is available in [Section 13.5: Peak-tracking and extended sinmux](#).

The peak-tracking feature is automatically reset by the device when the monitoring window is closed. Anyway, it is possible to perform the reset by the host interface by setting the PT_INIT bit of the EMB_FUNC_INIT_B (67h) register to 1. This must be done in case the host has to disable one FSM while the peak-tracking feature is still running.

7 FIFO

The LSM6DSV320X finite state machine allows storing FSM events or peak-tracking output data in the FIFO buffer. This is particularly useful when the host enters a sleep or power-down state in which it cannot read and process IMU data. Keeping the LSM6DSV320X device powered on allows running edge processing, whose outputs can be retrieved by the host the next time it enters the run state.

Details about the LSM6DSV320X FIFO configuration are available in AN6119.

7.1 FSM events in FIFO

The LSM6DSV320X finite state machine supports storing FSM events in the FIFO buffer by setting the `FSM_FIFO_EN` bit of the `EMB_FUNC_FIFO_EN_B` (45h) embedded functions register to 1.

The FSM program can store the following FSM events in the FIFO buffer:

- Events related to the long counter timeout, which are stored in FIFO every time the long counter value reaches the configured timeout value.
- Events related to the long counter value, which are stored in FIFO every time a specific SETM command, needed to store the long counter value, is issued. Refer to the dedicated paragraph for more details on its usage.
- Events related to the detected motion pattern, which are stored in FIFO every time a `OUTC / CONT / CONTREL` command is issued (so when the FSM writes its `OUTSx` registers or sets its `IS_FSMx` bit of the `FSM_STATUS_MAINPAGE` (4Ah) register or `FSM_STATUS` (13h) embedded functions register).

All the above cases are identified by a FIFO tag value readable from the `FIFO_DATA_OUT_TAG` (78h) register equal to 1Fh, but they can be distinguished by decoding the values of the `FIFO_DATA_OUT_X_L` (79h) and `FIFO_DATA_OUT_X_H` (7Ah) registers. The `FIFO_DATA_OUT_Y_L` (7Bh), `FIFO_DATA_OUT_Y_H` (7Ch), `FIFO_DATA_OUT_Z_L` (7Dh) and `FIFO_DATA_OUT_Z_H` (7Eh) registers contain the timestamp value corresponding to the generated FSM event. The format of the FSM events in FIFO is indicated in the following table.

Table 6. Finite state machine events in FIFO

Event	<code>FIFO_DATA_OUT_TAG[7:3]</code> (78h)	<code>FIFO_DATA_OUT_X_L</code> (79h)	<code>FIFO_DATA_OUT_X_H</code> (7Ah)	<code>FIFO_DATA_OUT_Y_L</code> (7Bh)	<code>FIFO_DATA_OUT_Y_H</code> (7Ch)	<code>FIFO_DATA_OUT_Z_L</code> (7Dh)	<code>FIFO_DATA_OUT_Z_H</code> (7Eh)
Long counter timeout	0x1F	0x00	0x00	TIMESTAMP0	TIMESTAMP1	TIMESTAMP2	TIMESTAMP3
<code>OUTSx / IS_FSMx</code>	0x1F	<code>OUTSx</code>	Index of FSM ⁽¹⁾	TIMESTAMP0	TIMESTAMP1	TIMESTAMP2	TIMESTAMP3
Long counter value	0x1F	<code>FSM_LONG_COUNTER_L</code> (48h)	0x09	TIMESTAMP0	TIMESTAMP1	TIMESTAMP2	TIMESTAMP3
	0x1F	<code>FSM_LONG_COUNTER_H</code> (49h)	0x0A	TIMESTAMP0	TIMESTAMP1	TIMESTAMP2	TIMESTAMP3

1. FSMs are indexed from 1 to 8.

The high part of the long counter value, identified by a `FIFO_DATA_OUT_X_H` (7Ah) register value equal to 0Ah, is stored in FIFO only if the `FSM_LONG_COUNTER_H` (49h) register value is not zero.

7.2 Peak-tracking outputs in FIFO

The LSM6DSV320X finite state machine supports storing the peak-tracking output data in the FIFO buffer every time a specific SETM command, needed to stop the peak-tracking monitoring, is issued. Refer to [Section 10.2.8: SETM \(F9h\)](#) for more details on its usage.

The peak-tracking output data are identified by a FIFO tag value readable from the FIFO_DATA_OUT_TAG (78h) register equal to 18h. The FIFO_DATA_OUT_X_L (79h), FIFO_DATA_OUT_X_H (7Ah), FIFO_DATA_OUT_Y_L (7Bh), FIFO_DATA_OUT_Y_H (7Ch), FIFO_DATA_OUT_Z_L (7Dh), and FIFO_DATA_OUT_Z_H (7Eh) registers contain the X, Y, and Z high-g accelerometer data in half-precision floating-point format. The format of peak-tracking outputs in FIFO is indicated in the following table.

Table 7. Format of peak-tracking output data

Event	FIFO_DATA_OUT_TAG[7:3] (78h)	FIFO_DATA_OUT_X_L (79h)	FIFO_DATA_OUT_X_H (7Ah)	FIFO_DATA_OUT_Y_L (7Bh)	FIFO_DATA_OUT_Y_H (7Ch)	FIFO_DATA_OUT_Z_L (7Dh)	FIFO_DATA_OUT_Z_H (7Eh)
Peak-tracking output data	0x18	X ⁽¹⁾		Y ⁽¹⁾		Z ⁽¹⁾	

1. High-g accelerometer data in half-precision floating-point format.

8 Fixed Data Section

The **[Fixed Data Section]** stores information about the **[Variable Data Section]** and the **[Instructions Section]**. It is composed of six bytes and it is located at the beginning of each program. The following figure shows the structure of the **[Fixed Data Section]**.

Figure 11. [Fixed Data Section]

	NAME	7	6	5	4	3	2	1	0			
0	CONFIG A	NR_THRESH(1:0)		NR_MASK(1:0)		NR_LTIMER(1:0)		NR_TIMER(1:0)				
1	CONFIG B	DES	EXT_SINMUX	ANGLE	PAS	DECTREE	STOPDONE	0	JMP			
2	SIZE	PROGRAM SIZE(7:0)										
3	SETTINGS	MASKSEL(1:0)		SIGNED	R_TAM	THRS3SEL	IN_SEL(2:0)					
4	RESET POINTER	RESET POINTER(7:0)										
5	PROGRAM POINTER	PROGRAM POINTER(7:0)										

Note: The green colored bits have to be set according to program purposes, while the red colored bits have to be set to 0 when the program is loaded into the embedded advanced features registers and are handled by the FSM program. The white colored bits must be written to 0 for the correct operation of the device.

The first two bytes store the amount of resources used by the program, while other bytes are used by the device to store the program status.

- With CONFIG_A it is possible to declare:
 - Up to three thresholds (NR_THRESH bits)
 - Up to three masks (NR_MASK bits)
 - Up to two long (16 bits) timers (NR_LTIMER bits)
 - Up to two short (8 bits) timers (NR_TIMER bits)
- With CONFIG_B it is possible to declare:
 - A decimation factor for incoming ODR (DES bit)
 - An extended sinmux capability, used to select as FSM input, the long counter value or the value of a filter/feature computed by the machine learning core (EXT_SINMUX bit)
 - Usage of gyroscope angles that have to be computed and stored (ANGLE bit)
 - Usage of previous axis signs that have to be computed and stored (PAS bit)
 - Usage of a decision tree interface (DECTREE bit)
- The SIZE parameter stores the length in bytes of the whole program (sum of **[Fixed Data Section]** size, **[Variable Data Section]** size and Instruction section size). The SIZE byte must always be an even number. If the size of the program is odd, an additional STOP state has to be added at the bottom of the Instruction section.
- The SETTINGS parameter stores the current program status (selected mask, selected threshold, input signal, and so forth).
- The RESET POINTER (RP) and PROGRAM POINTER (PP) store respectively the reset pointer relative address (jump address when a RESET condition is true) and the program pointer relative address (address of the instruction under execution during the current sample time). Address 00h is referred to CONFIG_A byte.

Note: When PP is equal to 0, the device automatically runs the start routine (refer to [Section 12: Start routine for additional information](#)) in order to properly initialize the internal variables and parameters of the state machine. This is mandatory for a correct operation of the device.

9 Variable Data Section

The **[Variable Data Section]** is located below the corresponding **[Fixed Data Section]** of a program, and its size depends on the amount of resources defined in the **[Fixed Data Section]**.

Each resource enumerated in the **[Fixed Data Section]** is then allocated in the **[Variable Data Section]**, with the proper size and at the proper position. The following figure shows the structure of the **[Variable Data Section]**.

Figure 12. [Variable Data Section]

	NAME	7	6	5	4	3	2	1	0
6	THRESH1	THRESH1(15:0)							
7									
8	THRESH2	THRESH2(15:0)							
9									
10	THRESH3	THRESH3(15:0)							
11									
12	EXT_SINMUX	SINMUX_ADDR_B				SINMUX_ADDR_A			
13		IN_SEL(3)	THRXYZ1	NEXT_ODR	0	SINMUX_ADDR_C			
14	MASKA	MASKA(7:0)							
15	TMASKA	TMASKA(7:0)							
16	MASKB	MASKB(7:0)							
17	TMASKB	TMASKB(7:0)							
18	MASKC	MASKC(7:0)							
19	TMASKC	TMASKC(7:0)							
20	DELTAT	DELTAT(15:0)							
21									
22	DX	DX(15:0)							
23									
24	DY	DY(15:0)							
25									
26	DZ	DZ(15:0)							
27									
28	DV	DV(15:0)							
29									
30	TC	TC(15:0) or TC(7:0)							
31									
32	TIMER1	TIMER1(15:0)							
33									
34	TIMER2	TIMER2(15:0)							
35									
36	TIMER3	TIMER3(7:0)							
37	TIMER4	TIMER4(7:0)							
38	DEST	DEST(7:0)							
39	DESC	DESC(7:0)							
40	PAS	SCTC	CANGLE	MSKIT	MSKITEQ	SIGN_X	SIGN_Y	SIGN_Z	SIGN_V
41	DECTREE	0	0	DTSEL(1:0)			DTRES(3:0)		

Note: The orange colored bits have to be set according to program purposes, while the black colored bits must be set to 0 when the program is loaded into the embedded advanced features registers and are handled by the FSM program. The white colored bits must be set to 0 for the correct operation of the device.

As shown in the table above, the maximum size of the **[Variable Data Section]** is 36 bytes. If the program requires fewer resources, the size allocated for the **[Variable Data Section]** is lower. Bytes from 0 to 5, not shown in the table above, are allocated for the CONFIG A, CONFIG B, SIZE, SETTINGS, RP, and PP bytes of the **[Fixed Data Section]**.

Note: The usage of the resources declared in the **[Fixed Data Section]** starts always from the lowest resource number. For example if the user defines `NR_THRESH = 10` in the **[Fixed Data Section]** (two thresholds defined), the available thresholds that can be used in the program are `THRESH1` and `THRESH2`, while `THRESH3` is not available and the bytes corresponding to `THRESH3` are not allocated (all the resources below `THRESH2` are shifted up).

9.1 Thresholds

Threshold resources are used to check and validate values assumed by the selected input signal (through the SINMUX command) and axis (through MASKS) in comparison conditions.

The unit of measurement of the threshold is that of the selected signal:

- If the LSM6DSV320X low-*g* accelerometer signal is selected, the unit of the threshold is [*g*].
- If the LSM6DSV320X high-*g* accelerometer signal is selected, the unit of the threshold is [*hg*].
- If the LSM6DSV320X gyroscope signal is selected, the unit of the threshold is [*rad/sec*].
- If the LSM6DSV320X integrated gyroscope signal is selected, the unit of the threshold is [*rad*].
- If the external sensor signal is selected, the unit is the same used after applying the sensitivity.
- If the long counter value is selected, the threshold is expressed in 15-bit unsigned format.
- If the MLC filtered signal or computed feature is selected, the unit of the threshold is the same as that of the selected filtered signal or the selected computed feature.

Thresholds can be signed or unsigned. It is possible to move from signed to unsigned mode by using the SSIGN0 / SSIGN1 commands. In signed mode, the signal and threshold keep their original sign in the comparison. In unsigned mode, the comparison is performed between the absolute values of both signal and threshold.

By setting the NR_THRESH[1:0] bits of CONFIG_A byte, the corresponding number of thresholds can be configured in the **[Variable Data Section]**, as described below:

- NR_THRESH[1:0] = 00: no thresholds are allocated in the **[Variable Data Section]**.
- NR_THRESH[1:0] = 01: only THRESH1[15:0] is allocated in the **[Variable Data Section]**.
- NR_THRESH[1:0] = 10: THRESH1[15:0] and THRESH2[15:0] are allocated in the **[Variable Data Section]**.
- NR_THRESH[1:0] = 11: THRESH1[15:0], THRESH2[15:0] and THRESH3[15:0] are allocated in the **[Variable Data Section]**.

Involved commands:

- STHR1 / STHR2
- SELTHR1 / SELTHR3
- SSIGN0 / SSIGN1

Involved conditions:

- GNTH1 / GNTH2 / GLTH1 / GRTH1
- LNTH1 / LNTH2 / LLTH1 / LRTH1

9.2 Extended sinmux

The extended sinmux resource is used to select as FSM inputs the long counter value (SINMUX with argument 08h), the high-*g* accelerometer data (SINMUX with first argument 09h), or a feature / filter computed by the machine learning core (SINMUX with first argument 09h). In addition, this resource is also required when there is the need for executing multiple conditions in one ODR (THRXYZ1).

By setting the EXT_SINMUX bit of CONFIG_B byte to 1, the EXT_SINMUX bytes are allocated in the **[Variable Data Section]** (the EXT_SINMUX bytes values are automatically managed by the device). This is mandatory if at least one of the commands listed below is expected to be used in the program.

Involved commands:

- SINMUX with first argument equal to 08h (long counter selection) or 09h (high-*g* accelerometer data or MLC feature / filter selection)
- THRXYZ0, THRXYZ1

Involved conditions:

- N/A

9.3 Masks / temporary masks

Mask resources are used to enable or disable mask action on the input data (X, Y, Z, V) when a condition is performed. If a mask bit is set to 1, then the corresponding axis and sign is enabled, otherwise it is disabled. In case the input data is generated by an external sensor with 1-axis in 3-byte data format, its data is available in the X channel of the masks. Masks are used in threshold comparison conditions or zero-crossing detection. Masks allow inverting the sign of the input signal by enabling the corresponding axis bit with a minus sign. Masks are composed of 8 bits (2 bits for each axis), as shown below:

+X	-X	+Y	-Y	+Z	-Z	+V	-V
----	----	----	----	----	----	----	----

For each axis, it is possible to configure four different mask settings:

1. Positive axis bit = 0 / negative axis bit = 0, axis is disabled.
2. Positive axis bit = 0 / negative axis bit = 1, axis with opposite sign is enabled.
3. Positive axis bit = 1 / negative axis bit = 0, axis with current sign is enabled.
4. Positive axis bit = 1 / negative axis bit = 1, axis with current sign and axis with opposite sign are enabled.

When a program is enabled, the value of each mask is copied inside the related temporary mask (TM), which is used during execution of conditions. Each time a condition is issued, the result of the condition is stored again in the temporary mask (it affects also consecutive conditions).

Example:

- GNTH1 condition
- THRESH1 = 0.50 g
- MASKA = 12h (00010010b) → -Y and +V are enabled
- Current input accelerometer sample = [0.72 -0.45 0.77 1.15]

TM before the condition	0	0	0	1	0	0	1	0
Accelerometer sample	0.72	-0.72	-0.45	0.45	0.77	-0.77	1.15	-1.15
TM after the condition	0	0	0	0	0	0	1	0

It is possible to reset the temporary mask value to the mask value in the following conditions:

- Anytime there is a reset condition
- When executing a CONTREL command
- When executing a REL command
- After each true next condition, if an SRTAM1 command has been previously issued

By setting the NR_MASK[1:0] bits of CONFIG_A byte, the corresponding number of masks can be configured in the [Variable Data Section], as described below:

- NR_MASK[1:0] = 00: no masks are allocated in the [Variable Data Section].
- NR_MASK[1:0] = 01: only MASKA[7:0]/TMASKA[7:0] are allocated in the [Variable Data Section].
- NR_MASK[1:0] = 10: MASKA[7:0]/TMASKA[7:0] and MASKB[7:0]/TMASKB[7:0] are allocated in the [Variable Data Section].
- NR_MASK[1:0] = 11: MASKA[7:0]/TMASKA[7:0], MASKB[7:0]/TMASKB[7:0], and MASKC[7:0]/TMASKC[7:0] are allocated in the [Variable Data Section].

Involved commands:

- SELMA / SELMB / SELMC
- SMA / SMB / SMC
- REL
- SRTAM0 / SRTAM1

Involved conditions:

- GNTH1 / GNTH2 / GLTH1 / GRTH1
- LNTH1 / LNTH2 / LLTH1 / LRTH1
- PZC / NZC

9.4 Angle calculation

Angle resources can be used instead of angular velocity data when a condition is issued. The angle computation is performed internally: gyroscope data are automatically multiplied by the DELTAT value, and the results are added to corresponding angle axis bytes (DX, DY, DZ and DV). This feature is enabled when the ANGLE bit of the CONFIG B byte of the **[Fixed Data Section]** is set to 1. The integration is performed each time a new sample is processed by the FSM, independently of the selected signal.

There are two reset-angle modalities:

- By default, angular velocity integration is cleared each time a reset or next condition is true. In this case, computed angles (DX, DY, DZ and DV bytes) restart from zero when a new sample arrives.
- If the program contains a CANGLE command, a different reset-angle modality is used. In this case, integrated angles are cleared:
 - If a CANGLE command is performed (when a new sample arrives)
 - Only if a reset condition is true

By setting the ANGLE bit of the CONFIG_B byte to 1, 10 bytes (DELTAT, DX, DY, DZ and DV) are allocated in the **[Variable Data Section]**: DELTAT resource has to be set equal to current FSM_ODR cycle time in seconds (half floating point (16 bits) format). If a CANGLE command is expected to be used, also the PAS bit of the CONFIG_B byte has to be set to 1.

Involved commands:

- CANGLE

Involved conditions:

- GNTH1 / GNTH2 / GLTH1 / GRTH1
- LNTH1 / LNTH2 / LLTH1 / LRTH1
- PZC / NZC

9.5 TC and timers

Timer resources are used to manage event durations. It is possible to declare two kinds of timer resources: long timers (16 bits) and short timers (8 bits). The time base is set by the FSM_ODR[2:0] bits of the FSM_ODR (5Fh) register, including the decimation factor if used. Long timer resources are called TI1 and TI2, while short timer resources are called TI3 and TI4. An additional internal timer counter (TC) is used as temporary counter to check if a timer has elapsed. The TC value can be preloaded with two different modalities, selectable by using the SCTC0 / SCTC1 commands:

- SCTC0 mode (default): when the program pointer moves to a state with a timeout condition, the TC value is always preloaded to the corresponding timer value. In this modality, the timer duration affects one state only.
- SCTC1 mode: when the program pointer moves to a state with a timeout condition, there are two different scenarios depending on which timer is used in the new state:
 - If the timer used in the new state is different from the timer used in the previous state, the TC value is preloaded to the corresponding timer value. In this modality, the timer duration affects one state only (same as SCTC0 mode).
 - If the timer used in the new state is the same used in the previous state, the TC value is not preloaded. The TC value continues to be decreased starting from its previous value. In this modality, the timer duration could affect more states.

The TC value is decreased by 1 each time a new sample occurs. If TC reaches 0, the condition is true.

Example:

- Timer TI3 is set equal to 10 samples. Consider the following states:
 - S0 - SCTC0 or SCTC1
 - S1 - TI3 | GNTH1
 - S2 - TI3 | LNTH2
 - S3 - TI3 | GNTH1
- TI3 = 0Ah (10 samples)

Depending on S0, there are two different state machine behaviors:

- SCTC0 case: the TC byte is always preloaded (when the program pointer moves to states S1, S2, and S3) and each condition is checked for a maximum of 10 samples. This means that all conditions can be verified in a maximum of 30 samples.
- SCTC1 case: the TC byte is preloaded only when the program pointer moves to S1 (and is not preloaded when it moves to S2 and S3), and all conditions have to be verified in a maximum of 10 samples.

SCTC1 modality is typically used when different conditions have to be verified in the same time window.

By setting the NR_LTIMER[1:0] bits of the CONFIG_A byte, the corresponding number of long timers can be configured in the **[Variable Data Section]**, as described below:

- NR_LTIMER[1:0] = 00: no long timers are allocated in the **[Variable Data Section]**.
- NR_LTIMER[1:0] = 01: TIMER1[15:0] is allocated in the **[Variable Data Section]**
- NR_LTIMER[1:0] = 10: TIMER1[15:0] and TIMER2[15:0] are allocated in the **[Variable Data Section]**.

By setting the NR_TIMER[1:0] bits of the CONFIG_A byte, the corresponding number of short timers can be configured in the **[Variable Data Section]**, as described below:

- NR_TIMER[1:0] = 00: no short timers are allocated in the **[Variable Data Section]**.
- NR_TIMER[1:0] = 01: TIMER3[7:0] is allocated in the **[Variable Data Section]**.
- NR_TIMER[1:0] = 10: TIMER3[7:0] and TIMER4[7:0] are allocated in the **[Variable Data Section]**.

Below the size of the TC resource:

- If NR_LTIMER[1:0] = 00 and NR_TIMER[1:0] = 00, the TC resource is not allocated.
- If NR_LTIMER[1:0] = 00 and NR_TIMER[1:0] ≠ 00, the TC resource occupies one byte.
- If NR_LTIMER[1:0] ≠ 00 and NR_TIMER[1:0] = 00, the TC resource occupies two bytes.
- If NR_LTIMER[1:0] ≠ 00 and NR_TIMER[1:0] ≠ 00, the TC resource occupies two bytes.

Involved commands:

- STIMER3 / STIMER4
- SCTC0 / SCTC1

Involved conditions:

- TI1 / TI2 / TI3 / TI4

9.6 Decimator

The decimator resource is used to reduce the sample rate of the data going to the finite state machine.

By setting the DES bit of the CONFIG_B byte to 1, the DEST and DESC bytes can be properly configured in the **[Variable Data Section]**. The DEST value is the desired decimation factor, while the DESC value is the internal counter (automatically managed by the device). The decimation factor is related to the FSM_ODR[2:0] bits of the FSM_ODR (5Fh) register, according to the following formula:

$$\text{PROGRAM_ODR} = \text{FSM_ODR} / \text{DEST}$$

At startup:

$$\text{DESC} = \text{DEST} \text{ (initial decimation value)}$$

When the sample clock occurs:

$$\text{DESC} = \text{DESC} - 1$$

When DESC is equal to 0, the current sample is used as the new input for the state machine, and the DESC value is set to the initial decimation value again.

Involved commands:

- N/A

Involved conditions:

- N/A

Note: Since a DEST value equal to 0 is invalid and a DEST value equal to 1 means "no decimation", the minimum meaningful DEST value is 2.

9.7 Previous axis sign

The previous axis sign resource is mainly used to store the sign of the previous sample: this information is used in zero-crossing conditions. In addition, it is also used to store other information such as the selected timer reset method (SCTC0 or SCTC1), the clear angle flag (CANGLE) used to reset the integrated gyroscope data (DX, DY, DZ, and DV bytes inside the **[Variable Data Section]**) and the selected interrupt mask type (MSKIT, MSKITEQ, or UMSKIT).

By setting the PAS bit of the CONFIG_B byte to 1, the PAS byte is allocated in the **[Variable Data Section]** (the PAS byte value is automatically managed by the device). This is mandatory if at least one of the commands or conditions listed below is expected to be used in the program.

Involved commands:

- SCTC0 / SCTC1 / CANGLE / MSKIT / MSKITEQ / UMSKIT

Involved conditions:

- PZC / NZC

Note: If the SSIGN0 command is performed, NZC and PZC are used as a generic ZC condition.

9.8 MLC interface

The MLC interface of the FSM includes the possibility of implementing conditions on the output of a decision tree or on the value of a computed filter / feature. This can be very useful when a machine learning logic or a custom filter is expected to be combined with an FSM program.

The output of a decision tree is accessible by using the CHKDT condition, which can be used to evaluate the result of one of the four decision trees available inside the machine learning core algorithms.

By setting the DECTREE bit of CONFIG_B byte to 1, the DECTREE byte can be properly configured in the **[Variable Data Section]**. The DECTREE byte contains information about the progressive number of the decision trees to be triggered (DTSEL[1:0] bits, from 0 to 3) and the corresponding expected value (DTRES[3:0] bits, from 0 to 15).

The value of a filter / feature computed by the MLC can be selected using the extended sinmux feature.

By setting the EXT_SINMUX bit of the CONFIG_B byte to 1, the EXT_SINMUX bytes are allocated in the **[Variable Data Section]** (the EXT_SINMUX bytes are automatically managed by the device). Refer to [Section 9.2: Extended sinmux](#) and [Section 10.2.22: SINMUX \(23h\)](#) for more details about how to select a filter / feature computed by the MLC.

Note: Using the SETP command allows reconfiguring dynamically the DECTREE byte inside the program flow in order to trigger a different decision tree and its expected value. Details about the SETP command are provided in its dedicated paragraph.

Note: Refer to AN6292 for more details about how to configure the MLC.

Involved commands:

- SINMUX

Involved conditions:

- CHKDT

10 Instructions Section

The [Instructions Section] is defined below the [Variable Data Section] and is composed of a series of states that implement the algorithm logic. Each state is characterized by one 8-bit operation code (opcode), and each opcode can implement a command or a RESET/NEXT condition:

- Commands are used to perform special tasks for flow control, output, and synchronization. Some commands may have parameters, executed as one single-step command.
- RESET/NEXT conditions are a combination of two conditions (4 bits for RESET condition and 4 bits for NEXT condition) that are used to reset or continue the program flow.

The opcodes have a direct effect on registers and internal state machine memories. For some opcodes, additional side effects can occur (such as update of status information).

A RESET/NEXT condition or a command, eventually followed by parameters, represents an instruction, also called program state. They are the building blocks of the instructions section of a program.

10.1 RESET/NEXT conditions

The RESET/NEXT conditions are used to reset or continue the program flow, and are composed of one single state.

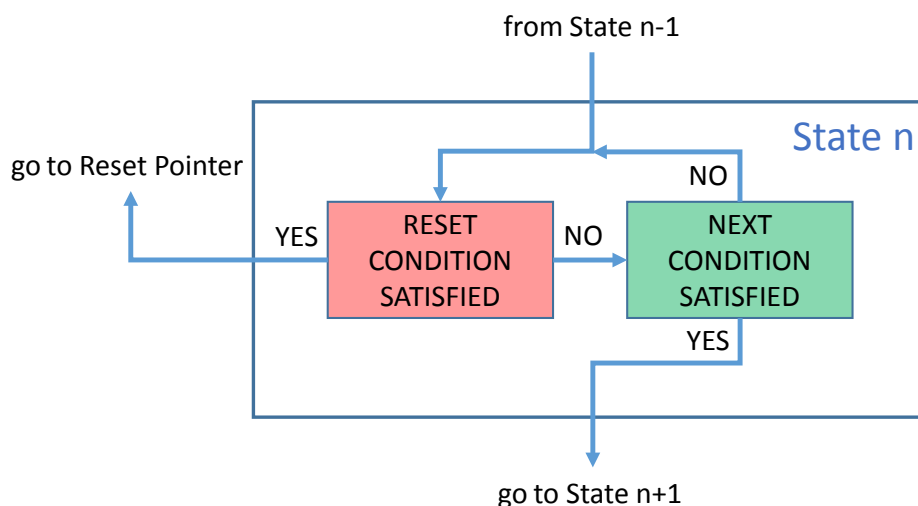
The RESET condition is defined in the opcode MSB part while the NEXT condition is defined in the opcode LSB part.

The RESET/NEXT conditions affect the program flow as indicated below:

- A transition to the reset pointer occurs whenever the RESET condition is true ($PP = RP$).
- A transition to the next state occurs whenever the RESET condition is false and the NEXT condition is true ($PP = PP + 1$).
- No transitions occur when both the RESET and NEXT conditions are false.

As shown in the following figure, the RESET condition is always performed before the NEXT condition that is evaluated only when the RESET condition is not satisfied.

Figure 13. Single state description



By default, a single RESET/NEXT condition is executed when a new data sample is processed by the FSM, but it is possible to execute multiple conditions on the same data sample by using the THRXYZ1/THRXYZ0 commands. Details about how to enable this mode can be found in [Section 10.2.28: THRXYZ1 \(F7h\)](#) and [Section 10.2.29: THRXYZ0 \(F8h\)](#).

Note: The RESET condition is always evaluated before the NEXT condition. By default, the reset pointer (RP) is set to the first state, but it is possible to dynamically change the reset pointer (RP) by using SRP/CRP commands.

Since a condition is coded over four bits, a maximum of 16 different conditions can be coded: the list of available conditions is shown in the following table. There are four types of conditions:

- Timeouts: these conditions are true when the TC counter, preloaded with a timer value, reaches zero.
- Threshold comparisons: these conditions are true when enabled inputs such as the accelerometer X, Y, Z axis or norm are higher (or lower) than a programmed threshold.

$$V = \sqrt{x^2 + y^2 + z^2}$$
- Zero-crossing detection: these conditions are true when an enabled input crosses the zero level.
- Decision tree output check: these conditions are true when the selected decision tree output is equal to the desired value (see [Section 9.8: MLC interface](#) for more details).

Table 8. Conditions

OP code	Mnemonic	Description	Note	Resources needed
0h	NOP	No operation	Execution moves to another condition	N/A
1h	TI1	Timer 1 (16-bit value) valid	No evaluation of data samples	TC, TIMER1
2h	TI2	Timer 2 (16-bit value) valid		TC, TIMER1, TIMER2
3h	TI3	Timer 3 (8-bit value) valid		TC, TIMER3
4h	TI4	Timer 4 (8-bit value) valid		TC, TIMER3, TIMER4
5h	GNTH1	Any triggered axis \geq THRESH1	Input signal, triggered with mask, compared to threshold	THRESH1, one MASK
6h	GNTH2	Any triggered axis \geq THRESH2		THRESH1, THRESH2, one MASK
7h	LNTH1	Any triggered axis $<$ THRESH1		THRESH1, one MASK
8h	LNTH2	Any triggered axis $<$ THRESH2		THRESH1, THRESH2, one MASK
9h	GLTH1	All triggered axes \geq THRESH1		THRESH1, one MASK
Ah	LLTH1	All triggered axes $<$ THRESH1		THRESH1, one MASK
Bh	GRTH1	Any triggered axis \geq -THRESH1		THRESH1, one MASK
Ch	LRTH1	Any triggered axis $<$ -THRESH1		THRESH1, one MASK
Dh	PZC	Any triggered axis crossed zero value, with positive slope	Input signal, triggered with mask, crossing zero value	PAS
Eh	NZC	Any triggered axis crossed zero value, with negative slope		PAS
Fh	CHKDT	Check result from a decision tree vs. expected	Requires machine learning core configuration	DECTREE

The last column of the table above indicates the resource needed by the conditions. These resources are allocated inside the **[Variable Data Section]** and can be different between one FSM and another. For correct FSM behavior, it is mandatory to set the amount of resources needed by each program in the **[Fixed Data Section]**.

Note: Having the same condition in the NEXT and RESET positions does not make sense. Consequently, opcodes such as 11h do not implement the TI1 | TI1 condition, but implement some commands: for example, the opcode 11h implements the CONT command.

Moreover, it is not possible to perform the following conditions because they are recognized as commands:

- PZC | CHKDT opcode (0xDF) is equal to SMB opcode.
- NZC | CHKDT opcode (0xEF) is equal to MSKITEQ opcode.
- CHKDT | GNTH1 opcode (0xF5) is equal to MSKIT opcode.
- CHKDT | LNTH1 opcode (0xF7) is equal to THRXYZ1 opcode.
- CHKDT | GNTH2 opcode (0xF6) is equal to RSTLC opcode.
- CHKDT | LNTH2 opcode (0xF8) is equal to THRXYZ0 opcode.
- CHKDT | GLTH1 opcode (0xF9) is equal to SETM opcode.
- CHKDT | PZC opcode (0xFD) is equal to DECR opcode.
- CHKDT | NZC opcode (0xFE) is equal to SMC opcode.

10.1.1 NOP (0h)

Description: NOP (no operation) is used as filler for the RESET/NEXT pair for some particular conditions, which do not need an active opposite condition.

Actions:

- If NOP is in the RESET condition, the FSM evaluates only the NEXT condition.
- If NOP is in the NEXT condition, the FSM evaluates only the RESET condition.

10.1.2 TI1 (1h)

Description: TI1 condition counts and evaluates the counter value of the TC bytes.

Action:

- When the program pointer moves to a state with a TI1 condition, $TC = \text{TIMER1}$.
- When a new data sample (X, Y, Z, V) is processed by the FSM, then $TC = TC - 1$:
 - If $TC > 0$, continue comparisons in the current state.
 - If $TC = 0$, the condition is valid:
 - If TI1 is in the RESET position, $PP = RP$.
 - If TI1 is in the NEXT position, $PP = PP + 1$.

Note: Details about how to use timers inside the THRXYZ1 / THRXYZ0 commands can be found in [Section 10.2.28: THRXYZ1 \(F7h\)](#) and [Section 10.2.29: THRXYZ0 \(F8h\)](#).

10.1.3 TI2 (2h)

Description: TI2 condition counts and evaluates the counter value of the TC bytes.

Action:

- When the program pointer moves to a state with a TI2 condition, $TC = \text{TIMER2}$.
- When a new data sample (X, Y, Z, V) is processed by the FSM, then $TC = TC - 1$:
 - If $TC > 0$, continue comparisons in the current state.
 - If $TC = 0$, the condition is valid:
 - If TI2 is in the RESET position, $PP = RP$.
 - If TI2 is in the NEXT position, $PP = PP + 1$.

Note: Details about how to use timers inside the THRXYZ1 / THRXYZ0 commands can be found in [Section 10.2.28: THRXYZ1 \(F7h\)](#) and [Section 10.2.29: THRXYZ0 \(F8h\)](#).

10.1.4 TI3 (3h)

Description: TI3 condition counts and evaluates the counter value of the TC byte.

Action:

- When the program pointer moves to a state with a TI3 condition, $TC = \text{TIMER3}$.
- When a new data sample (X, Y, Z, V) is processed by the FSM, then $TC = TC - 1$:
 - If $TC > 0$, continue comparisons in the current state.
 - If $TC = 0$, the condition is valid:
 - If TI3 is in the RESET position, $PP = RP$.
 - If TI3 is in the NEXT position, $PP = PP + 1$.

Note: Details about how to use timers inside the THRXYZ1 / THRXYZ0 commands can be found in [Section 10.2.28: THRXYZ1 \(F7h\)](#) and [Section 10.2.29: THRXYZ0 \(F8h\)](#).

10.1.5 TI4 (4h)

Description: TI4 condition counts and evaluates the counter value of the TC byte.

Action:

- When the program pointer moves to a state with a TI4 condition, $TC = \text{TIMER4}$.
- When a new data sample (X, Y, Z, V) is processed by the FSM, then $TC = TC - 1$:
 - If $TC > 0$, continue comparisons in the current state.
 - If $TC = 0$, the condition is valid:
 - If TI4 is in the RESET position, $PP = RP$.
 - If TI4 is in the NEXT position, $PP = PP + 1$.

Note: Details about how to use timers inside the **THRXYZ1** / **THRXYZ0** commands can be found in [Section 10.2.28: THRXYZ1 \(F7h\)](#) and [Section 10.2.29: THRXYZ0 \(F8h\)](#).

10.1.6 GNTH1 (5h)

Description: GNTH1 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is greater than or equal to the threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X \geq \text{threshold} || -X \geq \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If GNTH1 is valid and it is in the RESET position, $PP = RP$.
 - If GNTH1 is valid and it is in the NEXT position, $PP = PP + 1$.

10.1.7 GNTH2 (6h)

Description: GNTH2 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is greater than or equal to the threshold 2 level. The threshold used during the comparison is THRESH2.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X \geq \text{threshold} || -X \geq \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If GNTH2 is valid and it is in the RESET position, $PP = RP$.
 - If GNTH2 is valid and it is in the NEXT position, $PP = PP + 1$.

10.1.8 LNTH1 (7h)

Description: LNTH1 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is lower than the threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X < \text{threshold} || -X < \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If LNTH1 is valid and it is in the RESET position, $PP = RP$.
 - If LNTH1 is valid and it is in the NEXT position, $PP = PP + 1$.

10.1.9 LNTH2 (8h)

Description: LNTH2 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is lower than the threshold 2 level. The threshold used during the comparison is THRESH2.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X < \text{threshold} \parallel -X < \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If LNTH2 is valid and it is in the RESET position, PP = RP.
 - If LNTH2 is valid and it is in the NEXT position, PP = PP + 1.

10.1.10 GLTH1 (9h)

Description: GLTH1 condition is valid if all axes of the FSM processed data sample (X, Y, Z, V) are greater than or equal to the threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X \geq \text{threshold} \parallel -X \geq \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If GLTH1 is valid and it is in the RESET position, PP = RP.
 - If GLTH1 is valid and it is in the NEXT position, PP = PP + 1.

10.1.11 LLTH1 (Ah)

Description: LLTH1 condition is valid if all axes of the FSM processed data sample (X, Y, Z, V) are lower than the threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X < \text{threshold} \parallel -X < \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If LLTH1 is valid and it is in the RESET position, PP = RP.
 - If LLTH1 is valid and it is in the NEXT position, PP = PP + 1.

10.1.12 GRTH1 (Bh)

Description: GRTH1 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is greater than or equal to the reversed threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is -THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is -THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X \geq \text{threshold} \parallel -X \geq \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If GRTH1 is valid and it is in the RESET position, PP = RP.
 - If GRTH1 is valid and it is in the NEXT position, PP = PP + 1.

10.1.13 LRTH1 (Ch)

Description: LRTH1 condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) is lower than the reversed threshold 1 level. The threshold used during the comparison depends on the THRS3SEL bit in the SETTINGS byte of the **[Fixed Data Section]** as described below:

- THRS3SEL = 0 (default value or value assumed after the SELTHR1 command is performed): the threshold used is -THRESH1.
- THRS3SEL = 1 (value assumed after the SELTHR3 command is performed): the threshold used is -THRESH3.

Note: In case the "+" and "-" signs of the same axis are enabled, it is enough that one of them satisfies the condition, which means applying an OR operator (for example, $+X < \text{threshold} \parallel -X < \text{threshold}$).

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If LRTH1 is valid and it is in the RESET position, PP = RP.
 - If LRTH1 is valid and it is in the NEXT position, PP = PP + 1.

10.1.14 PZC (Dh)

Description: PZC condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) crossed the zero level, with a positive slope.

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If a zero-crossing event with positive slope occurs and PZC is in the RESET position, PP = RP.
 - If a zero-crossing event with positive slope occurs and PZC is in the NEXT position, PP = PP + 1.

10.1.15 NZC (Eh)

Description: NZC condition is valid if any triggered axis of the FSM processed data sample (X, Y, Z, V) crossed the zero level, with a negative slope.

Action:

- When a new sample set (X, Y, Z, V) occurs, check the condition:
 - If a zero-crossing event with negative slope occurs and NZC is in the RESET position, PP = RP.
 - If a zero-crossing event with negative slope occurs and NZC is in the NEXT position, PP = PP + 1.

10.1.16 CHKDT (Fh)

Description: CHKDT condition is valid if the result of the selected decision tree is the expected one. For additional information about how to properly configure the decision tree Interface refer to [Section 9.8: MLC interface](#).

Action:

- When a new sample set (X, Y, Z, V) occurs, then check the output of the selected decision tree; if the output is the expected one:
 - If CHKDT is in the RESET position, $PP = RP$.
 - If CHKDT is in the NEXT position, $PP = PP + 1$.

10.2 Commands

Commands are used to modify the program behavior in terms of flow control, output, and synchronization. Commands are immediately executed (no need for a new sample set). When a command is executed, the program pointer is set to the next line, which is immediately evaluated.

Some commands may need parameters that must be defined (through dedicated opcodes reporting the parameter value) just below the command opcode. Refer to the example below that shows three consecutive opcodes used to dynamically change the value of the “THRESH1” resource when the STHR1 command is executed:

- AAh (STHR1 command)
- CDh (1st parameter)
- 3Ch (2nd parameter)

When the program pointer reaches the AAh (STHR1 command) state, the device recognizes that this is a command which requires two parameters: these three states are immediately executed without waiting for a new sample set. After the command execution is completed, the THRESH1 resource value is set to 3CCDh, equal to 1.2.

Table 9. List of commands

Opcode	Mnemonic	Description	Parameter
00h	STOP	Stop execution, and wait for a new start from reset pointer	None
11h	CONT	Continues execution from reset pointer	None
22h	CONTREL	Continues execution from reset pointer, resetting temporary mask	None
33h	SRP	Set reset pointer to next address/state	None
44h	CRP	Clear reset pointer to first program line	None
55h	SETP	Set parameter in program memory	Byte 1: address Byte 2: value
B5h	SETR	Set device register value (ASC)	Byte 1: address (ASC) Byte 2: value (ASC)
F9h	SETM	Set device embedded advanced features register value	Byte 1: embedded advanced features register page Byte 2: embedded advanced features register address Byte 3: bitmask Byte 4: value
66h	SELMA	Select MASKA and TMASKA as current mask	None
77h	SELMB	Select MASKB and TMASKB as current mask	None
88h	SELMC	Select MASKC and TMASKC as current mask	None
99h	OUTC	Write the temporary mask to output registers	None
AAh	STHR1	Set new value to THRESH1 register	Byte 1: THRESH1 [LSB] Byte 2: THRESH1 [MSB]
BBh	STHR2	Set new value to THRESH2 register	Byte 1: THRESH2 [LSB] Byte 2: THRESH2 [MSB]
CCh	SELTHR1	Selects THRESH1 instead of THRESH3	None
DDh	SELTHR3	Selects THRESH3 instead of THRESH1	None
FFh	REL	Reset temporary mask to default	None
12h	SSIGN0	Set UNSIGNED comparison mode	None
13h	SSIGN1	Set SIGNED comparison mode	None
14h	SRTAM0	Do not reset temporary mask after a next condition true	None
21h	SRTAM1	Reset temporary mask after a next condition true	None
23h	SINMUX	Set input multiplexer	Refer to Section 10.2.22: SINMUX (23h) for details about the parameters that are needed.

Opcode	Mnemonic	Description	Parameter
24h	STIMER3	Set new value to TIMER3 register	Byte 1: TI3 value
31h	STIMER4	Set new value to TIMER4 register	Byte 1: TI4 value
34h	INCR	Increase long counter +1 and check long counter timeout	None
FDh	DECR	Decrease long counter -1	None
F6h	RSTLC	Reset long counter	None
F7h	THRXYZ1	Enable execution of multiple conditions without waiting for a sample set	None
F8h	THRXYZ0	Disable execution of multiple conditions without waiting for a sample set	None
41h	JMP	Jump address for two Next conditions	Byte 1: conditions Byte 2: reset jump address Byte 3: next jump address
42h	CANGLE	Clear angle	
43h	SMA	Set MASKA and TMASKA	Byte 1: MASKA value
DFh	SMB	Set MASKB and TMASKB	Byte 1: MASKB value
FEh	SMC	Set MASKC and TMASKC	Byte 1: MASKC value
5Bh	SCTC0	Clear time counter TC on next condition true	None
7Ch	SCTC1	Do not clear time counter TC on next condition true	None
C7h	UMSKIT	Unmask interrupt generation when setting OUTS	None
EFh	MSKITEQ	Mask interrupt generation when setting OUTS if OUTS does not change	None
F5h	MSKIT	Mask interrupt generation when setting OUTS	None

10.2.1 STOP (00h)

Description: STOP command halts execution and waits for host restart. This command is used to control the end of the program.

Parameters: none

Actions:

- Outputs the resulting mask to OUTS_x register
- Generates interrupt (if enabled, according to the use of MSKIT / MSKITEQ / UMSKIT commands)
- Stops itself by setting the STOPDONE bit in the CONFIG_B byte of the **[Fixed Data Section]** to 1. The user should disable and enable the corresponding state machine bit in the FSM_ENABLE (46h) register to restart the program. In this case, the start routine is performed. For additional information about the start routine refer to [Section 12: Start routine](#).

10.2.2 CONT (11h)

Description: CONT command loops execution to the reset point. This command is used to control the end of the program.

Parameters: none

Actions:

- Outputs the resulting mask to the OUTS_x registers
- Generates interrupt (if enabled, accordingly with use of MSKIT / MSKITEQ / UMSKIT commands)
- PP = RP

10.2.3 **CONTREL (22h)**

Description: CONTREL command loops execution to the reset point. This command is used to control the end of the program. In addition, it resets the temporary mask value to its default value.

Parameters: none

Actions:

- Outputs the resulting mask to the OUTS_x registers
- Resets temporary mask to default value
- Generates interrupt (if enabled, according to the use of MSKIT / MSKITEQ / UMSKIT commands)
- $PP = RP$

10.2.4 **SRP (33h)**

Description: SRP command sets the reset pointer to the next address/state. This command is used to modify the starting point of the program.

Parameters: none

Actions:

- $RP = PP + 1$
- $PP = PP + 1$

10.2.5 **CRP (44h)**

Description: CRP command clears the reset pointer to the start position (at the beginning of the program code).

Parameters: none

Actions:

- $RP = \text{beginning of program code}$
- $PP = PP + 1$

10.2.6 **SETP (55h)**

Description: SETP command allows the configuration of the state machine currently used to be modified. This command is used to modify a byte value at a desired address of the current state machine.

Parameters: two bytes

- 1st parameter: address (8 bits) of the byte to be modified. This address is relative to the current state machine (address 00h refers to CONFIG_A byte).
- 2nd parameter: new value (8 bits) to be written in the 1st parameter address

Actions:

- Byte value addressed by 1st parameter = 2nd parameter
- $PP = PP + 3$

10.2.7 SETR (B5h)

10.2.7.1 ASC feature

Description: The SETR command is used to enable the adaptive self-configuration (ASC) feature, which allows the device to reconfigure itself without any intervention from the host processor. The registers that can be written by the FSM are listed in the tables below.

Table 10. ASC FSM main page registers

Main page register	Register address (controller)	Register address (FSM)
FIFO_CTRL1	07h	07h
FIFO_CTRL2	08h	08h
FIFO_CTRL3	09h	09h
FIFO_CTRL4	0Ah	0Ah ⁽¹⁾
CTRL1	10h	10h
CTRL2	11h	11h
CTRL1_XL_HG	4Eh	13h
CTRL2_XL_HG	4Dh	14h
CTRL6	15h	15h
CTRL7	16h	16h
CTRL8	17h	17h
CTRL9	18h	18h
CTRL10	19h	19h

1. FIFO_MODE change not supported.

Table 11. ASC FSM embedded functions registers

Embedded functions register	Register address (controller)	Register address (FSM)
EMB_FUNC_EN_A	04h	01h ⁽¹⁾
EMB_FUNC_EN_B	05h	02h ⁽¹⁾
FSM_ENABLE	46h	03h ⁽¹⁾
EMB_FUNC_FIFO_EN_A	44h	05h ⁽¹⁾
EMB_FUNC_FIFO_EN_B	45h	06h ⁽¹⁾
EMB_FUNC_INIT_A	66h	08h ⁽¹⁾
EMB_FUNC_INIT_B	67h	0Ch ⁽¹⁾
SFLP_ODR	5Eh	0Dh ⁽¹⁾
FSM_ODR	5Fh	0Eh ⁽¹⁾
MLC_ODR	60h	0Fh ⁽¹⁾

1. Access to the embedded functions registers is automatically handled by the FSM.

Write access to the above device registers is mutually exclusive: the FSM_WR_CTRL_EN bit of the FUNC_CFG_ACCESS (01h) register is used to give write capability of the above device registers to either the controller or the FSM. After writing this bit, the controller change is confirmed through the assertion or deassertion of the FSM_WR_CTRL_STATUS bit of the CTRL_STATUS (1Ah) register as described below:

- FSM_WR_CTRL_STATUS equal to 0: all the device registers are writable from the standard interface only.
- FSM_WR_CTRL_STATUS equal to 1: the above device registers are under FSM control and are in read-only mode from the standard interface.

Parameters: two bytes

- 1st parameter: address (8 bits) of the register whose value is to be modified, referred to as the "Register address (FSM)" column of the above tables. If this parameter is equal to 0x00, the 2nd parameter is used as a write bitmask for the first subsequent SETR command.
- 2nd parameter: new value (8 bits) to be written in the 1st parameter register address. If the 1st parameter is 0x00, this parameter is used as a write bitmask for the first subsequent SETR command.

Actions:

- Register value addressed by 1st parameter = 2nd parameter
- PP = PP + 3

Note: *The default write bitmask is 0xFF, which means that all bits are written. Setting a bitmask allows changing the value of specific bits of a register without changing the value of the other bits. For example, if the accelerometer ODR must be changed without changing the accelerometer power mode, two consecutive SETR commands must be performed as described below:*

1. SETR 0x00 0x0F (bitmask equal to 0x0F)
2. SETR 0x10 0x06 (new accelerometer ODR equal to 120 Hz)

10.2.8 SETM (F9h)

Description: SETM command allows the FSM to configure the value of a desired embedded advanced features register.

Parameters: four bytes

- 1st parameter: embedded advanced features register page
- 2nd parameter: embedded advanced features register address
- 3rd parameter: bitmask
- 4th parameter: value

Actions:

- Byte value addressed by 1st and 2nd parameters = 4th parameter (with bit masking based on 3rd parameter)
- PP = PP + 5

10.2.8.1 Long counter value trigger in FIFO

The FSM can be configured to store the current long counter value in FIFO by issuing the SETM command below:

- SETM command: F9h
- 1st parameter: 01h
- 2nd parameter: 7Dh
- 3rd parameter: 80h
- 4th parameter: 80h

10.2.8.2 Peak tracking

The FSM can start or stop the peak-tracking monitoring window by issuing the SETM command below:

- SETM command: F9h
- 1st parameter: 01h
- 2nd parameter: 7Dh
- 3rd parameter: 01h
- 4th parameter: 01h (start) or 00h (stop)

Note: *The peak-tracking monitoring is executed at the FSM_ODR rate and after all the FSMs have been executed. It cannot be activated and deactivated in the same ODR cycle.*

10.2.9 SELMA (66h)

Description: SELMA command sets MASKA / TMASKA as current mask.

Parameters: none

Actions:

- MASK_A is selected. It sets the SETTINGS(MASKSEL[1:0]) bits of the **[Fixed Data Section]** to 00.
- PP = PP + 1

10.2.10 SELMB (77h)

Description: SELMB command sets MASKB / TMASKB as current mask.

Parameters: none

Actions:

- MASK_B is selected. It sets the SETTINGS(MASKSEL[1:0]) bits of the **[Fixed Data Section]** to 01.
- PP = PP + 1

10.2.11 SELMC (88h)

Description: SELMC command sets MASKC / TMASKC as current mask.

Parameters: none

Actions:

- MASK_C is selected. It sets the SETTINGS(MASKSEL[1:0]) bits of the **[Fixed Data Section]** to 10.
- PP = PP + 1

10.2.12 OUTC (99h)

Description: OUTC stands for output command. This command is used to update the OUTS register value to the current temporary mask value and to generate an interrupt (if enabled).

Parameters: none

Actions:

- Updates the OUTS register of the current state machine to the selected temporary mask value
- Generates interrupt (if enabled, according to the use of the MSKIT / MSKITEQ / UMSKIT commands)
- PP = PP + 1

10.2.13 STHR1 (AAh)

Description: STHR1 command sets the THRESH1 value to a new desired value. THRESH1 is a half floating-point (16 bits) number.

Parameters: two bytes

- 1st parameter: THRESH1 LSB value (8 bits)
- 2nd parameter: THRESH1 MSB value (8 bits)

Actions:

- Sets new value for THRESH1
- PP = PP + 3

10.2.14 STHR2 (BBh)

Description: STHR2 command sets the THRESH2 value to a new desired value. THRESH2 is a half floating-point (16 bits) number.

Parameters: two bytes

- 1st parameter: THRESH2 LSB value (8 bits)
- 2nd parameter: THRESH2 MSB value (8 bits)

Actions:

- Sets new value for THRESH2
- PP = PP + 3

10.2.15 SELTHR1 (CCh)

Description: after executing the SELTHR1 command, the THRESH1 value is used instead of the THRESH3 value when the GNTH1, LNTH1, GLTH1, LLTH1, GRTH1, LRTH1 conditions are performed.

Parameters: none

Actions:

- Selects THRESH1 instead of THRESH3. It sets the SETTINGS(THRS3SEL) bit of the **[Fixed Data Section]** to 0.
- $PP = PP + 1$

10.2.16 SELTHR3 (DDh)

Description: after executing the SELTHR3 command, the THRESH3 value is used instead of the THRESH1 value when the GNTH1, LNTH1, GLTH1, LLTH1, GRTH1, LRTH1 conditions are performed.

Parameters: none

Actions:

- Selects THRESH3 instead of THRESH1. It sets the SETTINGS(THRS3SEL) bit of the **[Fixed Data Section]** to 1.
- $PP = PP + 1$

10.2.17 REL (FFh)

Description: REL command releases the temporary axis mask information.

Parameters: none

Actions:

- Resets current temporary masks to the default value
- $PP = PP + 1$

10.2.18 SSIGN0 (12h)

Description: SSIGN0 command sets the comparison mode to "unsigned".

Parameters: none

Actions:

- Sets comparison mode to "unsigned". It sets the SETTINGS(SIGNED) bit of the **[Fixed Data Section]** to 0.
- $PP = PP + 1$

10.2.19 SSIGN1 (13h)

Description: SSIGN1 command sets the comparison mode to "signed" (default behavior).

Parameters: none

Actions:

- Sets comparison mode to "signed". It sets the SETTINGS(SIGNED) bit of the **[Fixed Data Section]** to 1.
- $PP = PP + 1$

10.2.20 SRTAM0 (14h)

Description: SRTAM0 command is used to preserve the temporary mask value when a NEXT condition is true (default behavior).

Parameters: none

Actions:

- Temporary axis mask value does not change after valid NEXT condition. It sets the SETTINGS(R_TAM) bit of the [Fixed Data Section] to 0.
- $PP = PP + 1$

10.2.21 SRTAM1 (21h)

Description: SRTAM1 command is used to reset the temporary mask when a NEXT condition is true.

Parameters: none

Actions:

- Temporary axis mask value is reset after valid NEXT condition. It sets the SETTINGS(R_TAM) bit of the [Fixed Data Section] to 1.
- $PP = PP + 1$

10.2.22 SINMUX (23h)

Description: SINMUX command is used to change the input source for the current state machine. If the SINMUX command is not performed, the accelerometer signal is automatically selected as the default input source.

The standard SINMUX command can also be used to select the MLC filtered data. For this purpose, the MLC filter structure has to be configured as below:

- The first MLC filter $[F_x F_y F_z F_v^{(2)}]$ has to be applied to the sensor axes.
- The second MLC filter $[0\ 0\ 0\ G_v^{(3)}]$ has to be applied to the sensor norm.
- The third MLC filter $[H_x H_y H_z H_v^{(2)}]$ has to be applied to the sensor axes.
- The fourth MLC filter $[0\ 0\ 0\ J_v^{(3)}]$ has to be applied to the sensor norm.

The above requirement for the order and type (axes or norm) of the MLC filters can be overcome by using the extended sinmux feature, which allows getting any of the MLC filtered data.

Note: In case the user just needs to apply two filters to the sensor axes (filters on the sensor norm are not needed), it is necessary to configure also the second MLC filter on the sensor norm even if it is not used. Furthermore, in case the user just needs to apply two filters to the sensor norm (filters on the sensor axes are not needed), it is necessary to configure all four MLC filters as described above.

In addition, the extended sinmux feature allows using the SINMUX command to select the long counter value or any computed MLC filter or feature.

Parameters: three bytes if the 1st parameter is equal to 09h, otherwise one byte.

- 1st parameter: value to select input source:
 - 00h: low-g accelerometer $[a_x a_y a_z a_v]$
 - 01h: gyroscope $[g_x g_y g_z g_v]$
 - 02h: external sensor $[m_x m_y m_z m_v]$
 - 03h: first filtered signal from the machine learning core⁽¹⁾ $[F_x F_y F_z F_v^{(2)}]$
 - 04h: third filtered signal from the machine learning core⁽¹⁾ $[H_x H_y H_z H_v^{(2)}]$
 - 05h: second filtered signal norm from the machine learning core⁽¹⁾ $[0\ 0\ 0\ G_v^{(3)}]$
 - 06h: fourth filtered signal norm from the machine learning core⁽¹⁾ $[0\ 0\ 0\ J_v^{(3)}]$
 - 07h: integrated gyroscope signal $[d_x d_y d_z d_v]$
 - 08h: long counter value $[LC_x\ 0\ 0\ 0]$
 - 09h: high-g accelerometer axes $[h_x h_y h_z h_v]$, high-g accelerometer norm $[0\ 0\ 0\ h_v]$, or any filter or feature from the machine learning core $[K_x K_y K_z K_v]$. The filter or the feature is selected through the 2nd and 3rd parameters
- 2nd parameter (needed only if the 1st parameter is equal to 09h):
 - 52h: high-g accelerometer axes
 - 5Ah: high-g accelerometer norm
 - XXh: MLC filter / feature IDENTIFIER[7:0]
- 3rd parameter (needed only if the 1st parameter is equal to 09h):
 - 01h: high-g accelerometer axes
 - 01h: high-g accelerometer norm
 - YYh: MLC filter / feature IDENTIFIER[15:8]

Identifiers for filters and features are indicated in the configuration file generated by STMicroelectronics tools when configuring the MLC as indicated in the figure below.

Figure 14. MLC identifiers for filters and features

```
-- <MLC1_SRC>DT1,0='negative',4='positive'

-- FILTER_IIR1_ACC_X -> 023Ch
-- FILTER_IIR1_ACC_Y -> 023Eh
-- FILTER_IIR1_ACC_Z -> 0240h
-- F1_MEAN_on_ACC_X -> 0242h
-- F2_MEAN_on_ACC_Y -> 0244h
-- F3_MEAN_on_ACC_Z -> 0246h
```

Note: The mask value to be used during a high-g accelerometer norm value comparison is 0x80 (corresponding to the +X axis) or 0x40 (corresponding to the -X axis).

Note: When a filter on the axes is intended to be selected through the SINMUX 09h command, it is recommended to configure the IDENTIFIER related to the filtered X-axis. In this case, the mask value during a comparison follows the normal order of the axes.

Note: The mask value to be used during a long counter value comparison is 0x80 (corresponding to +X axis), and the threshold is interpreted as an unsigned 15-bit value.

Note: The mask value to be used during a filter on the norm or a feature value comparison is 0x80 (corresponding to +X axis) or 0x40 (corresponding to -X axis).

Actions:

- Selects input signal accordingly with set parameter. It configures the SETTINGS(IN_SEL[2:0]) bits of the **[Fixed Data Section]** and the EXT_SINMUX(IN_SEL[3]) bit of the **[Variable Data Section]** according to the selected input source signal.
 - If the 1st parameter is equal to 9, PP + 4; otherwise, PP + 2.
- (1) Filter type could be HP / LP / IIR1 / IIR2 depending on the machine learning core configuration.
- (2) $F_v / H_v / K_v$ is internally computed by the FSM starting from $F_x, F_y, F_z / H_x, H_y, H_z / K_x, K_y, K_z$ filtered data values provided by the MLC.
- (3) G_v / J_v is provided by the MLC.

10.2.23 STIMER3 (24h)

Description: STIMER3 command is used to set a new value for TIMER3.

Parameters: one byte

- 1st parameter: new TIMER3 value

Actions:

- Sets new TIMER3 value
- PP = PP + 2

10.2.24 STIMER4 (31h)

Description: STIMER4 command is used to set a new value for TIMER4.

Parameters: one byte

- 1st parameter: new TIMER4 value

Actions:

- Sets new TIMER4 value
- PP = PP + 2

10.2.25 INCR (34h)

Description: INCR command is used to increase the long counter value by one. The long counter value is stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) embedded functions registers, and is clamped to the long counter timeout value stored into the FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh) embedded advanced features registers. An interrupt is generated when the long counter value is equal to the long counter timeout value. Details about the FSM long counter interrupt are available in [Section 4: FSM interrupt status and signal](#).

Parameters: none

Actions:

- Increase the long counter value by one, and generate an interrupt if the long counter value is equal to the long counter timeout value.
- $PP = PP + 1$

10.2.26 DECR (FDh)

Description: DECR command is used to decrease the long counter value by one. The long counter value is stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) embedded functions registers, and is clamped to zero.

Parameters: none

Actions:

- Decrease the long counter value by one.
- $PP = PP + 1$

10.2.27 RSTLC (F6h)

Description: RSTLC command is used to reset the long counter value. The long counter value is stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) embedded functions registers.

Parameters: none

Actions:

- Reset the long counter value.
- $PP = PP + 1$

10.2.28 THRXYZ1 (F7h)

Description: THRXYZ1 command is used to enable the mode where multiple conditions are executed on the same data sample. When this mode is enabled, the FSM executes all the conditions using the same data sample until a THRXYZ0 command is performed. All the instructions between the THRXYZ1 and THRXYZ0 commands can be considered as a single instruction. In this case, the commands and conditions affect the program flow as described below:

- If the current state is a command, it is immediately executed.
- If the current state is a condition:
 - If the RESET condition is true, the program pointer is set to the RP, and the THRXYZ1 bit of the EXT_SINMUX byte of the **[Variable Data Section]** is set to 0 to restore the default condition execution mode (refer to the THRXYZ0 command). The new state is executed when the next data sample is processed.
 - If the NEXT condition is true, the program pointer is set to the next state, which is immediately executed.
 - If the NEXT condition is false, the program pointer is set to the address of the THRXYZ1 command, and the FSM evaluates again the instructions between the THRXYZ1 and the THRXYZ0 commands when the next data sample is processed.

During the enablement of this mode, there are some considerations that must be considered.

- One timer only in SCTC1 mode can be used.
- If more than one OUTC command is issued before reaching the THRXYZ0 command, the OUTS register will contain only the information related to the result of the last performed condition.
- The JMP and SRP commands are not supported.

Parameters: none

Actions:

- Set the THRXYZ1 bit of the EXT_SINMUX byte of the **[Variable Data Section]** to 1.
- $PP = PP + 1$

10.2.29 THRXYZ0 (F8h)

Description: THRXYZ0 command is used to restore the default mode where each condition is executed on one data sample only. In this case, the commands and conditions affect the program flow as described below:

- If the current state is a command, it is immediately executed.
- If the current state is a condition:
 - If the RESET condition is true, the program pointer is set to the RP. The new state is executed when the next data sample is processed.
 - If the NEXT condition is true, the program pointer is set to the next state. The new state is executed when the next data sample is processed.
 - If both the RESET and NEXT conditions are false, the PP is not changed. This condition is evaluated again when the next data sample is processed.

Parameters: none

Actions:

- Set the THRXYZ1 bit of the EXT_SINMUX byte of the **[Variable Data Section]** to 0.
- $PP = PP + 1$

10.2.30 **JMP (41h)**

Description: JMP command is a special command characterized by a NEXT1 | NEXT2 condition, with two different jump addresses.

Parameters: three bytes

- 1st parameter: NEXT1 | NEXT2 condition
- 2nd parameter: jump address if NEXT1 condition is true
- 3rd parameter: jump address if NEXT2 condition is true

The NEXT1 condition is evaluated before the NEXT2 condition. Jump addresses are relative to the current state machine (address 00h refers to the CONFIG_A byte).

Actions:

- It sets to 1 the JMP bit in the CONFIG_B byte of the **[Fixed Data Section]**. Evaluates the NEXT1 | NEXT2 condition:
 - If the NEXT1 condition is true, PP = 2nd parameter address.
 - Else if the NEXT2 condition is true, PP = 3rd parameter address.
 - Else waits for a new sample set and evaluates again the NEXT1 | NEXT2 condition.

10.2.31 **CANGLE (42h)**

Description: CANGLE command is used to clear integrated gyroscope values. If this command is performed, integrated angle values are no longer cleared when a next condition is true (default behavior), except in the following cases:

- every time a CANGLE command is performed (when a new sample arrives)
- if a reset condition is true

Parameters: none

Actions:

- Clear angle values
- PP = PP + 1

10.2.32 **SMA (43h)**

Description: SMA command is used to set a new value for MASKA and TMASKA.

Parameters: one byte

- 1st parameter: new MASKA and TMASKA value

Actions:

- Set new MASKA and TMASKA value
- PP = PP + 2

10.2.33 **SMB (DFh)**

Description: SMB command is used to set a new value for MASKB and TMASKB.

Parameters: one byte

- 1st parameter: new MASKB and TMASKB value

Actions:

- Set new MASKB and TMASKB value
- PP = PP + 2

10.2.34 SMC (FEh)

Description: SMC command is used to set a new value for MASKC and TMASKC.

Parameters: one byte

- 1st parameter: new MASKC and TMASKC value

Actions:

- Set new MASKC and TMASKC value
- $PP = PP + 2$

10.2.35 SCTC0 (5Bh)

Description: SCTC0 command is used to reset the TC byte (time counter) when a NEXT condition is true (default behavior).

Parameters: none

Actions:

- TC (time counter) byte value is reset after valid NEXT condition.
- $PP = PP + 1$

10.2.36 SCTC1 (7Ch)

Description: SCTC1 command is used to preserve the TC byte (time counter) when a NEXT condition is true.

Parameters: none

Actions:

- TC (time counter) byte value does not change after valid NEXT condition.
- $PP = PP + 1$

10.2.37 UMSKIT (C7h)

Description: UMSKIT command is used to unmask interrupt generation when the OUTS register value is updated (default behavior). Refer to the OUTC / CONT / CONTREL commands for more details about interrupt generation.

Parameters: none

Actions:

- Unmask interrupt generation when setting the OUTS register.
- $PP = PP + 1$

10.2.38 MSKITEQ (EFh)

Description: MSKITEQ command is used to mask interrupt generation when the OUTS register value is updated but its value does not change (temporary mask value is equal to current OUTS register value). Refer to the OUTC / CONT / CONTREL commands for more details about interrupt generation.

Parameters: none

Actions:

- Mask interrupt generation when setting the OUTS register if OUTS does not change.
- $PP = PP + 1$

10.2.39 MSKIT (F5h)

Description: MSKIT command is used to mask interrupt generation when the OUTS register value is updated. Refer to the OUTC / CONT / CONTREL commands for more details about interrupt generation.

Parameters: none

Actions:

- Mask interrupt generation when setting the OUTS register.
- $PP = PP + 1$

11 FSM configuration example

This section contains an example that explains all write operations that have to be done in order to configure the LSM6DSV320X FSM. A few steps have to be followed:

- Configure the FSM registers inside the embedded functions registers set.
- Configure the FSM registers inside the embedded advanced features registers set.
- Configure the LSM6DSV320X sensor (accelerometer and / or gyroscope).

In this example, two simple programs are configured:

- PROGRAM 1: wrist tilt (around the x-axis) algorithm, routed to the INT1 pin
- PROGRAM 2: wake-up algorithm, routed to the INT2 pin

Both algorithms are intended to use accelerometer data only at a sample rate of 30 Hz.

Refer to the figure below for details about the [Program Data Section] and the [Instructions Section].

Figure 15. FSM configuration example

	PAGE - ADDRESS	NAME	7	6	5	4	3	2	1	0	
PROGRAM 1	4 - 00h	CONFIG A	01 (1 threshold)		01 (1 mask)		00		01 (1 short timer)		
	4 - 01h	CONFIG B	0	0	0	0	0	0	0	0	
	4 - 02h	SIZE	10h (16 bytes)								
	4 - 03h	SETTINGS	00		0	0	0	00			
	4 - 04h	RESET POINTER	00h								
	4 - 05h	PROGRAM POINTER	00h								
	4 - 06h	THRESH1	B7AEh (-0.480)								
	4 - 07h										
	4 - 08h	MASKA	80h (+X)								
	4 - 09h	TMASKA	00h								
	4 - 0Ah	TC	00h								
	4 - 0Bh	TIMER3	10h (16 samples)								
	4 - 0Ch	GNTH1 TI3	53h								
	4 - 0Dh	OUTC	99h								
	4 - 0Eh	GNTH1 NOP	50h								
	4 - 0Fh	STOP	00h								
PROGRAM 2	4 - 10h	CONFIG A	01 (1 threshold)		01 (1 mask)		00		00		
	4 - 11h	CONFIG B	0	0	0	0	0	0	0	0	
	4 - 12h	SIZE	0Ch (12 bytes)								
	4 - 13h	SETTINGS	00		0	0	0	00			
	4 - 14h	RESET POINTER	00h								
	4 - 15h	PROGRAM POINTER	00h								
	4 - 16h	THRESH1	3C66h (1.100)								
	4 - 17h										
	4 - 18h	MASKA	02h (+V)								
	4 - 19h	TMASKA	00h								
	4 - 1Ah	NOP GNTH1	05h								
	4 - 1Bh	CONTR EL	22h								

The FSM configuration has to be performed with both accelerometer and gyroscope sensors in power-down mode. Refer to the following script for the complete device configuration:

1. Write 00h to register 10h // Set accelerometer sensor in power-down mode
2. Write 00h to register 11h // Set gyroscope sensor in power-down mode
3. Write 80h to register 01h // Enable access to embedded functions registers
4. Write 01h to register 05h // EMB_FUNC_EN_B(FSM_EN) = 1
5. Write 4Bh to register 5Fh // FSM_ODR[2:0] = 001 (30 Hz)

```

6. Write 03h to register 46h // FSM_ENABLE = 03h
7. Write 01h to register 0Bh // FSM_INT1 = 01h
8. Write 02h to register 0Fh // FSM_INT2 = 02h
9. Write 40h to register 17h // PAGE_RW: enable write operation
10. Write 11h to register 02h // Enable access to embedded advanced features registers, PAGE_SEL = 1
11. Write 7Ah to register 08h // PAGE_ADDRESS = 7Ah
12. Write 00h to register 09h // Write 00h to register FSM_LONG_COUNTER_L
13. Write 00h to register 09h // Write 00h to register FSM_LONG_COUNTER_H
14. Write 02h to register 09h // Write 02h to register FSM_PROGRAMS
15. Write 7Eh to register 08h // PAGE_ADDRESS = 7Eh
16. Write 00h to register 09h // Write 00h to register FSM_START_ADDRESS_L
17. Write 04h to register 09h // Write 04h to register FSM_START_ADDRESS_H
18. Write 41h to register 02h // PAGE_SEL = 4
19. Write 00h to register 08h // PAGE_ADDRESS = 00h
20. Write 51h to register 09h // CONFIG_A
21. Write 00h to register 09h // CONFIG_B
22. Write 10h to register 09h // SIZE
23. Write 00h to register 09h // SETTINGS
24. Write 00h to register 09h // RESET POINTER
25. Write 00h to register 09h // PROGRAM POINTER
26. Write AEh to register 09h // THRESH1 LSB
27. Write B7h to register 09h // THRESH1 MSB
28. Write 80h to register 09h // MASKA
29. Write 00h to register 09h // TMASKA
30. Write 00h to register 09h // TC
31. Write 10h to register 09h // TIMER3
32. Write 53h to register 09h // GNTH1 | TI3
33. Write 99h to register 09h // OUTC
34. Write 50h to register 09h // GNTH1 | NOP
35. Write 00h to register 09h // STOP (mandatory for having even SIZE bytes)
36. Write 50h to register 09h // CONFIG_A
37. Write 00h to register 09h // CONFIG_B
38. Write 0Ch to register 09h // SIZE
39. Write 00h to register 09h // SETTINGS
40. Write 00h to register 09h // RESET POINTER
41. Write 00h to register 09h // PROGRAM POINTER
42. Write 66h to register 09h // THRESH1 LSB
43. Write 3Ch to register 09h // THRESH1 MSB
44. Write 02h to register 09h // MASKA
45. Write 00h to register 09h // TMASKA
46. Write 05h to register 09h // NOP | GNTH1
47. Write 22h to register 09h // CONTREL
48. Write 01h to register 02h // Disable access to embedded advanced features registers, PAGE_SEL = 0
49. Write 00h to register 17h // PAGE_RW: disable write operation

```

- 50. Write 00h to register 01h // Disable access to embedded functions registers
- 51. Write 02h to register 5Eh // MD1_CFG(INT1_EMB_FUNC) = 1
- 52. Write 02h to register 5Fh // MD2_CFG(INT2_EMB_FUNC) = 1
- 53. Write 04h to register 10h // CTRL1 = 04h (30 Hz)

12 Start routine

When the FSM is enabled, a start routine is automatically executed. The routine is executed if the PROGRAM POINTER byte is set to 0 and it performs the following tasks:

- The STOPDONE and the JMP bits in the CONFIG_B byte are reset.
- The PP and RP pointers are initialized to the first line of code.
- The SETTINGS field is initialized with the default value 0x20, which means:
 - MASKSEL = 00
 - SIGNED = 1
 - R_TAM = 0
 - THRS3SEL = 0
 - IN_SEL = 000
- The associated output register OUTS is cleared.
- Assign to all declared temporary masks the value of the corresponding original mask ($TMASK_x = MASK_x$).
- If timers are declared, the time counter is initialized to 0 ($TC = 0$).
- If decimation is declared, the decimation counter is initialized with the programmed decimation time value ($DESC = DEST$).
- If the previous axis sign resource is declared, it is initialized to 0 ($PAS = 0$).
- If the gyroscope angle computation is declared, the four angles are initialized to 0 ($DX = DY = DZ = DV = 0$).

When the start routine is performed, the program always restarts from a known state, independently of the way it was stopped. However, it should be noted that the default mode implies:

- MASKA selected as running mask (MASKSEL = 00).
- Signed comparison mode (SIGNED = 1).
- Do not release the temporary mask after a next condition is true (R_TAM = 0).
- Threshold1 selected instead of threshold3 for comparisons (THRS3SEL = 0).
- Input multiplexer set to select accelerometer data (IN_SEL = 000).

13 Examples of state machine configurations

13.1 Toggle

Toggle is a simple state machine configuration that generates an interrupt every n sample. The idea is to use a timer to count n samples.

Figure 16. Toggle state machine example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	00		00		00		01 (1 short timer)	
01h	CONFIG B	0	0	0	0	0	0	0	0
02h	SIZE	0Ah (10 bytes)							
03h	SETTINGS	00		0	0	0	00		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	TC	00h							
07h	TIMER3	10h (16 samples)							
08h	NOP TI3	03h							
09h	CONTREL	22h							

Instructions section description

PP = 08h: the first time this state is reached, TC = TI3. Each time a new sample set is generated, the TC byte is decreased by one. When TC = 0, PP = PP + 1.

PP = 09h: CONTREL command is executed without needing a sample set. This generates an interrupt and resets the program (PP = RP = 08h).

In the example, the interrupt is generated every 16 samples. TI3 can be configured in order to get the desired toggle period, which depends on the configured FSM_ODR.

13.2 Adaptive self-configuration (ASC)

This example shows how to use the ASC feature to reconfigure the device based on a specific motion event. The program below starts configuring the accelerometer in low-power mode at 30 Hz and the gyroscope in power-down mode. When a wake-up event is detected, both the sensors are configured in high-performance mode at 120 Hz. If stationary for a while, the accelerometer is set back to low-power mode at 30 Hz and the gyroscope is set back to power-down mode.

Figure 17. ASC state machine example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	01 (1 threshold)		01 (1 mask)		0		01 (1 short timer)	
01h	CONFIG B	0	0	0	1	0	0	0	0
02h	SIZE	20h (32 bytes)							
03h	SETTINGS	0		0	0	0	0		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	THRESH1	3C66h (1.100)							
07h									
08h	MASKA	02h (+v)							
09h	TMASKA	00h							
0Ah	TC	00h							
0Bh	TIMER3	78h (120 samples)							
0Ch	PAS	00h							
0Dh	MSKIT	F5h							
0Eh	SETR	B5h							
0Fh	10h	10h							
10h	63h	63h							
11h	SETR	B5h							
12h	11h	11h							
13h	00h	10h							
14h	NOP GNTH1	05h							
15h	SETR	B5h							
16h	10h	10h							
17h	06h	06h							
18h	SETR	B5h							
19h	11h	11h							
1Ah	06h	06h							
1Bh	SRP	33h							
1Ch	GNTH1 TI3	53h							
1Dh	CRP	44h							
1Eh	CONTREL	22h							
1Fh	STOP	00h							

Instructions section description

PP = 0Dh: MSKIT command is executed without needing a sample set. The MSKIT bit in the PAS byte is set to 1. PP = PP + 1.

PP = 0Eh: SETR command is executed without needing a sample set. The register 10h is set to 63h (the accelerometer sensor is configured in low-power mode at 30 Hz). PP = PP + 3.

PP = 11h: SETR command is executed without needing a sample set. The register 11h is set to 00h (the gyroscope sensor is configured in power-down mode). PP = PP + 3.

PP = 14h: this condition is evaluated each time a new sample is generated. If the vector (magnitude) of the acceleration signal is greater than THRESH1, PP = PP + 1.

PP = 15h: SETR command is executed without needing a sample set. The register 10h is set to 06h (the accelerometer sensor is configured in high-performance power mode at 120 Hz). PP = PP + 3.

PP = 18h: SETR command is executed without needing a sample set. The register 11h is set to 06h (the gyroscope sensor is configured in high-performance mode at 120 Hz). PP = PP + 3.

PP = 1Bh: SRP command is executed without needing a sample set. The RESET POINTER is set to the next state, 1Ch. PP = PP + 1.

PP = 1Ch: this condition is evaluated each time a new sample is generated. If the vector (magnitude) of the acceleration signal is greater than THRESH1, then PP = RP, and TC is set to TI3. If the vector (magnitude) of the acceleration signal is lower than THRESH1 for TI3 consecutive samples, PP = PP + 1.

PP = 1Dh: CRP command is executed without needing a sample set. The RESET POINTER is set to its default value, 0Dh. $PP = PP + 1$.

PP = 1Eh: CONTREL command is executed without needing a sample set. This does not generate an interrupt due to the execution of the MSKIT command and resets the program. $PP = RP = 0Dh$.

In the example, the wake-up threshold is 1.1 g, and the timer is 120 samples (which means 1 s).

13.3

Free-fall

This feature is used to detect when a system is dropping (for example, to protect data on the hard drive). If the object is in free-fall, the acceleration on the X-axis, Y-axis, and Z-axis goes to zero.

To implement this function, acceleration on all axes should be less than a configured threshold, for a minimum configured duration. When this condition is detected, an interrupt is generated.

Figure 18. Free-fall state machine example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	01 (1 threshold)		01 (1 mask)		00		01 (1 short timer)	
01h	CONFIG B	0	0	0	0	0	0	0	0
02h	SIZE	12h (18 bytes)							
03h	SETTINGS	00		0	0	0	00		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	THRESH1	34CDh (0.300)							
07h									
08h									
09h									
0Ah	MASKA	A8h (+X, +Y, +Z)							
09h	TMASKA	00h							
0Ah	TC	00h							
0Bh	TIMER3	03h (3 samples)							
0Ch	SSIGN0	12h							
0Dh	SRP	33h							
0Eh	GNTH1 TI3	53h							
0Fh	OUTC	99h							
10h	GNTH1 NOP	50h							
11h	STOP	00h							

Instructions section description

PP = 0Ch: SSIGN0 command is executed without needing a sample set. The SIGNED bit of the SETTINGS byte is set to 0, indicating that unsigned comparison mode was set. PP = PP + 1.

PP = 0Dh: SRP command is executed without needing a sample set. The RESET POINTER is set to the next state, 0Eh. PP = PP + 1.

PP = 0Eh: if the acceleration on one axis is greater than THRESH1, then PP = RP. If acceleration on all axes is lower than THRESH1 for three consecutive samples, then the PP is increased (PP = PP + 1).

PP = 0Fh: OUTC command is executed without needing a sample set. This generates an interrupt and increases the PP (PP = PP + 1).

PP = 10h: if acceleration on one axis is greater than THRESH1, then PP = RP. This means that the device is no longer in free-fall, so the program has to be reset.

In the example, the free-fall threshold is set to 0.3 g and the free-fall duration is set to three samples.

Note:

Free-fall duration is strictly related to FSM_ODR. For example, if FSM_ODR is set to 30 Hz, the free-fall duration is ~100 ms (three samples at 30 Hz).

13.4 Decision tree interface

This example shows how to use the decision tree interface with the FSM. It is assumed that the machine learning core is configured as below:

- Decision tree number 0 (the first one) implements an activity recognition algorithm able to detect three user activities (classes): stationary, walking, and running.
- An output value is associated with each recognized activity:
 - Stationary output is 0.
 - Walking output is 1.
 - Running output is 2.
- The window length for the features calculation is 2 seconds (60 samples having an ODR equal to 30 Hz).

The FSM implements a simple wake-up algorithm that is enabled after the output of the decision tree is equal to stationary. In this case, if the device starts moving again, a wake-up interrupt is generated by the FSM.

Figure 19. Decision tree interface example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	01 (1 threshold)		01 (1 mask)		00		10 (2 short timer)	
01h	CONFIG B	0	0	0	0	1	0	0	0
02h	SIZE	12h (18 bytes)							
03h	SETTINGS	00		0	0	0	00		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	THRESH1	3C33h (1.050)							
07h									
08h									
09h									
0Ah	TC	00h							
0Bh	TIMER3	02h (2 samples)							
0Ch	TIMER4	3Dh (61 samples)							
0Dh	DECTREE	00h (selected decision tree number 0, expected output is 0)							
0Eh	NOP CHKDT	0Fh							
0Fh	TI3 GNTH1	35h							
10h	OUTC	99h							
11h	TI4 NOP	40h							

Instructions section description

PP = 0Eh: check the decision tree output based on the DECTREE byte. The DECTREE byte is configured to check the decision tree number 0 and to expect an output equal to 0 (that is, stationary). If the detected activity is stationary, then the PP is increased ($PP = PP + 1$).

PP = 0Fh: if TI3 expires, then $PP = RP$ (the program is reset and the decision tree interface is checked again). If the vector (magnitude) of the accelerometer is greater than THRESH1, then PP is increased ($PP = PP + 1$).

PP = 10h: OUTC command is executed without needing a sample set. This generates an interrupt and increases the PP ($PP = PP + 1$).

PP = 11h: if TI4 expires, then $PP = RP$.

13.5 Peak-tracking and extended sinmux

This example shows how to use the peak tracking and the extended sinmux features for providing shock information to the host. This FSM implementation processes the high-g accelerometer data to generate two different interrupts corresponding to the start and the end of the shock event for allowing the host to compute the duration of the shock, and it uses the SETM command to store the peak-tracking outputs in the FIFO buffer.

In this example, a shock start event is generated when the high-g accelerometer norm signal is above 40 g, while the shock end event is generated when the high-g accelerometer norm signal returns below 39 g.

The FSM_FIFO_EN bit of the EMB_FUNC_FIFO_EN_B (45h) embedded functions register is considered set to 1, to allow the device to store the shock start and shock end events in FIFO.

Figure 20. Peak-tracking example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	10 (2 thresholds)		11 (3 masks)		0		0	
01h	CONFIG B	0	1	0	0	0	0	0	0
02h	SIZE	2Ah (42 bytes)							
03h	SETTINGS	0		0	0	0	0		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	THRESH1	3666h (0.400)							
07h									
08h	THRESH2	363Dh (0.390)							
09h									
0Ah	EXTENDED SINMUX	00h							
0Bh		00h							
0Ch	MASKA	80h (+X)							
0Dh	TMASKA	00h							
0Eh	MASKB	01h							
0Fh	TMASKB	00h							
10h	MASKC	00h							
11h	TMASKC	00h							
12h	SINMUX	23h							
13h	09h	09h							
14h	5Ah	5Ah							
15h	01h	01h							
16h	SELMA	66h							
17h	NOP LNTH2	08h							
18h	SETM	F9h							
19h	01h	01h							
1Ah	7Dh	7Dh							
1Bh	01h	01h							
1Ch	01h	01h							
1Dh	NOP GNTH1	05h							
1Eh	SELMB	77h							
1Fh	OUTC	99h							
20h	SELMA	66h							
21h	NOP LNTH2	08h							
22h	SETM	F9h							
23h	01h	01h							
24h	7Dh	7Dh							
25h	01h	01h							
26h	00h	00h							
27h	SELMC	88h							
28h	CONTREL	22h							
29h	STOP	00h							

Instructions section description

PP = 12h: SINMUX (extended) command is executed without needing a sample set. The high-g accelerometer norm signal is selected as the input of the FSM. PP = PP + 4.

PP = 16h: SELMA command is executed without needing a sample set. The MASKSEL(1:0) bits of the SETTINGS byte are set to 00b, indicating that the MASKA is selected. PP = PP + 1.

PP = 17h: this condition is evaluated each time a new sample is generated. If the vector (magnitude) of the high-g acceleration signal is lower than THRESH2, $PP = PP + 1$.

PP = 18h: SETM command is executed without needing a sample set. The peak-tracking monitoring window is started. $PP = PP + 5$.

PP = 1Dh: this condition is evaluated each time a new sample is generated. If the vector (magnitude) of the high-g acceleration signal is greater than THRESH1, $PP = PP + 1$.

PP = 1Eh: SELMB command is executed without needing a sample set. The MASKSEL(1:0) bits of the SETTINGS byte are set to 01b, indicating that the MASKB is selected. $PP = PP + 1$.

PP = 1Fh: OUTC command is executed without needing a sample set. Since the FSM_FIFO_EN bit is set to 1, the FSM shock start event is stored in FIFO. This generates an interrupt. $PP = PP + 1$.

PP = 20h: SELMA command is executed without needing a sample set. The MASKSEL(1:0) bits of the SETTINGS byte are set to 00b, indicating that the MASKA is selected. $PP = PP + 1$.

PP = 21h: this condition is evaluated each time a new sample is generated. If the vector (magnitude) of the high-g acceleration signal is lower than THRESH2, $PP = PP + 1$.

PP = 22h: SETM command is executed without needing a sample set. The peak-tracking monitoring window is ended. The peak-tracking outputs data are stored in FIFO. $PP = PP + 5$.

PP = 27h: SELMC command is executed without needing a sample set. The MASKSEL(1:0) bits of the SETTINGS byte are set to 10b, indicating that the MASKC is selected. $PP = PP + 1$.

PP = 28h: CONTREL command is executed without needing a sample set. Since the FSM_FIFO_EN bit is set to 1, the FSM shock end event is stored in FIFO. This generates an interrupt and resets the program ($PP = RP = 12h$).

14 Finite state machine tool

The finite state machine programmability in the device is allowed through a dedicated tool, available as an extension of MEMS Studio.

14.1 MEMS Studio

MEMS Studio is the graphical user interface for all the MEMS sensor demonstration boards available in the STMicroelectronics portfolio. It has the possibility to interact with a motherboard based on the STM32 microcontroller (professional MEMS tool), which enables the communication between the MEMS sensor and the PC GUI.

Details about the professional MEMS tool board can be found at [STEVAL-MKI109D](#).

MEMS Studio is available in three software packages for the three operating systems supported.

- Linux - [MEMS-Studio-Lin](#)
- Mac OS X - [MEMS-Studio-Mac](#)
- Windows - [MEMS-Studio-Win](#)

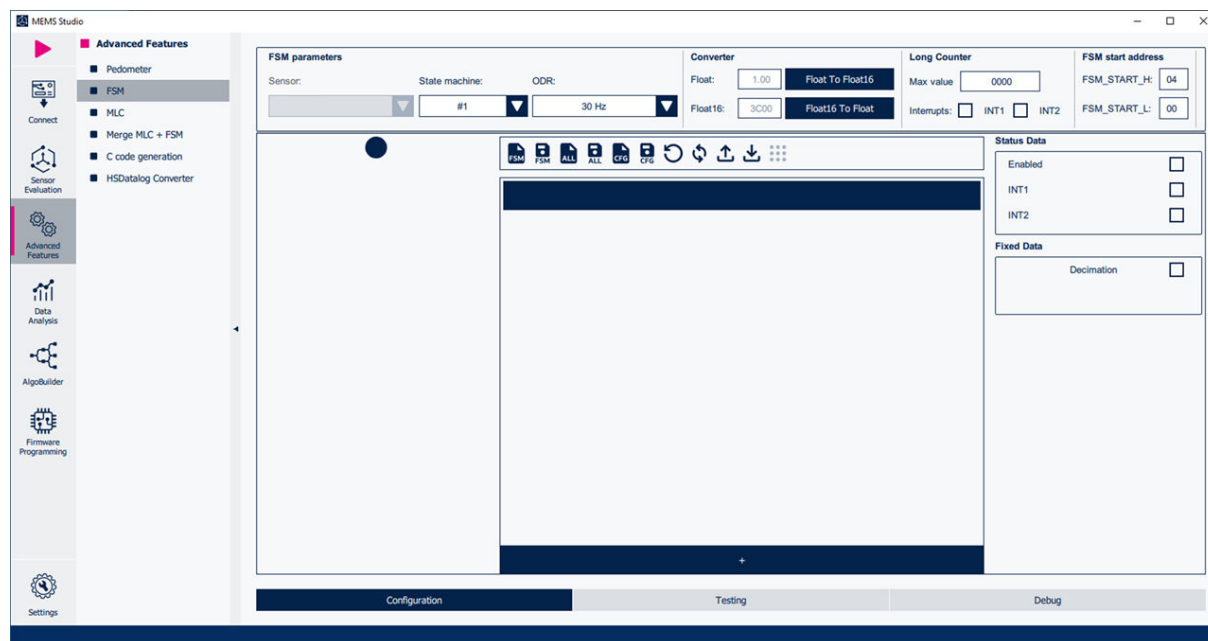
The MEMS Studio GUI allows visualization of sensor outputs in both graphical and numerical format and allows the user to save or generally manage data coming from the device.

MEMS Studio allows access to the MEMS sensor registers, enabling a fast prototype of register setup and easy test of the configuration directly in the device. It is possible to save the configuration of the current registers in a file and load a configuration from an existing file. This allows the sensor to be reprogrammed in a matter of seconds.

The finite state machine tool available in MEMS Studio helps the process of register configuration by automatically generating configuration files for the device. By clicking a few buttons, the configuration file is available. From these configuration files, the user can create his own library of configurations for the device.

To open the finite state machine programming tool, the user must first click the **[Advanced Features]** button on the left side of MEMS Studio and then the **[FSM]** button. When loaded, the **[Finite State Machine]** tool window is displayed as shown in the following figure.

Figure 21. Running the finite state machine tool



In the top part of the **[Finite State Machine]** tool main window, the user can select which state machine is selected (the selection is applied in both the **[Configuration]** tab and **[Debug]** tab). It is also possible to configure the FSM ODR and the long counter parameters. The FSM start address is automatically managed by the tool and should not be changed by the user.

The [Finite State Machine] tool is mainly composed of three tabs, which are detailed in dedicated sections:

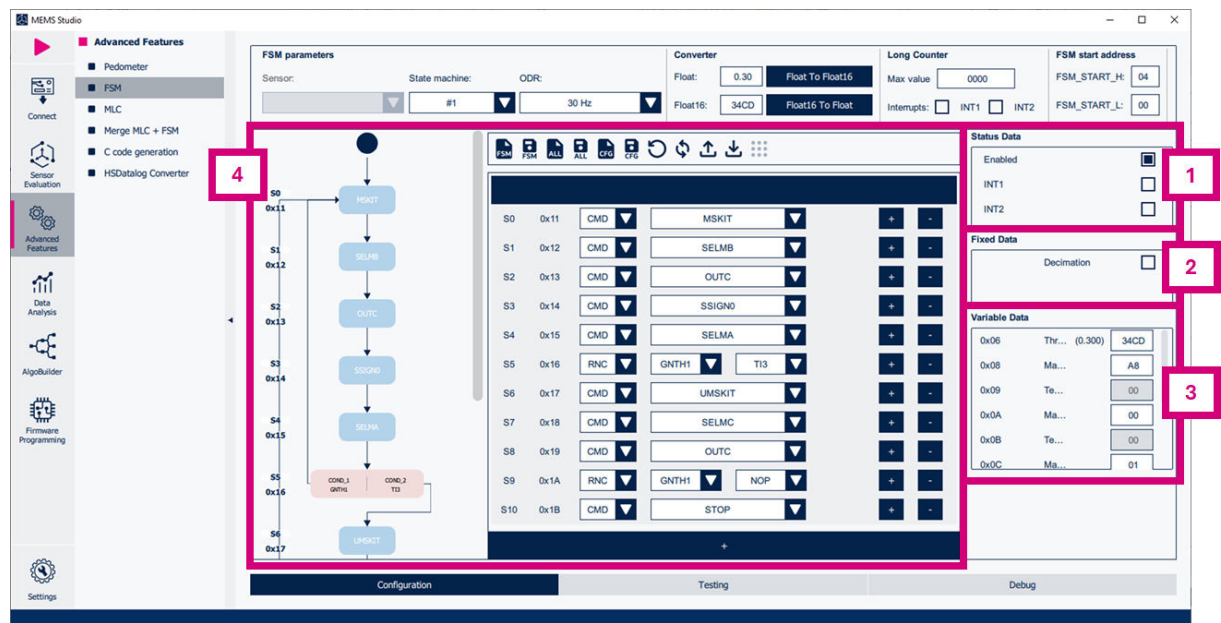
- [Configuration] tab (the one selected by default)
- [Testing] tab
- [Debug] tab

14.1.1 Configuration tab

The [Configuration] tab of the [Finite State Machine] tool allows the user to implement the program logic. The UI is able to abstract the FSM program structure: for this reason, four group boxes are shown:

1. [Status Data]
2. [Fixed Data]
3. [Variable Data]
4. [Instructions Section]

Figure 22. [Finite State Machine] tool - [Configuration] tab



In the top part of the [Instructions Section] tab, the user can manage the device configuration using dedicated buttons:

- [Load state machine from file]: it is used to load a .json file representing a single state machine in the selected state machine number.
- [Save state machine as]: it is used to save the selected state machine to a .json file that can be loaded in a second time using the [Load state machine from file] button.
- [Load all state machines from file]: it is used to load a .json file representing a set of state machines.
- [Save all state machine as]: it is used to save all the configured state machines to a .json file that can be loaded in a second time using the [Load all state machines from file] button.
- [Open device configuration from file]: it is used to load a .json file representing a configuration of the selected device and to perform the corresponding write transactions in the registers.
- [Save device configuration as]: it is used to save the current device configuration to a .json or .h file that can be loaded in a second time using the [Open device configuration from file] button.
- [Reset state machine]: it is used to clear the selected state machine.
- [Reset all state machines]: it is used to clear all the state machines.
- [Read FSM configuration from sensor]: it is used to read the device registers related to the FSM, and to graphically populate the UI based on the current FSM configuration and programs.
- [Write FSM configuration to sensor]: it is used to write the entire FSM configuration in the device (it includes FSM ODR, long counter parameters, interrupt status, and programs).

14.1.1.1 Status Data

The **[Status Data]** group box is available in the top-right corner of the **[Configuration]** tab.

Figure 23. [Configuration] tab - [Status Data]

Status Data	
Enabled	<input checked="" type="checkbox"/>
INT1	<input type="checkbox"/>
INT2	<input type="checkbox"/>

The **[Status Data]** group box allows the user to enable/disable the state machine and to route the interrupt status to the interrupt pins. In detail:

- The **[Enabled]** checkbox is used to enable/disable the state machine. It is automatically set if the program contains at least one instruction and it is automatically reset if the program does not contain any instruction.
- The **[INT1]** checkbox is used to enable routing the state machine interrupt to the INT1 pin.
- The **[INT2]** checkbox is used to enable routing the state machine interrupt to the INT2 pin.

If the device supports only one interrupt pin, MEMS Studio allows the user to route the state machine interrupt only to that pin.

14.1.1.2 Fixed Data

The **[Fixed Data]** group box is available on the right side of the **[Configuration]** tab.

Figure 24. [Configuration] tab - [Fixed Data]

Fixed Data	
Decimation	<input type="checkbox"/>

The **[Fixed Data]** group box allows the user to manually enable the resources that cannot be automatically enabled by analyzing the **[Instructions Section]**. All the other bits and bytes of the **[Fixed Data]** section are automatically managed by the tool based on the resources that are used in the **[Instructions Section]**.

14.1.1.3 Variable Data

The **[Variable Data]** group box is available on the right side of the **[Configuration]** tab.

Figure 25. [Configuration] tab – [Variable Data]

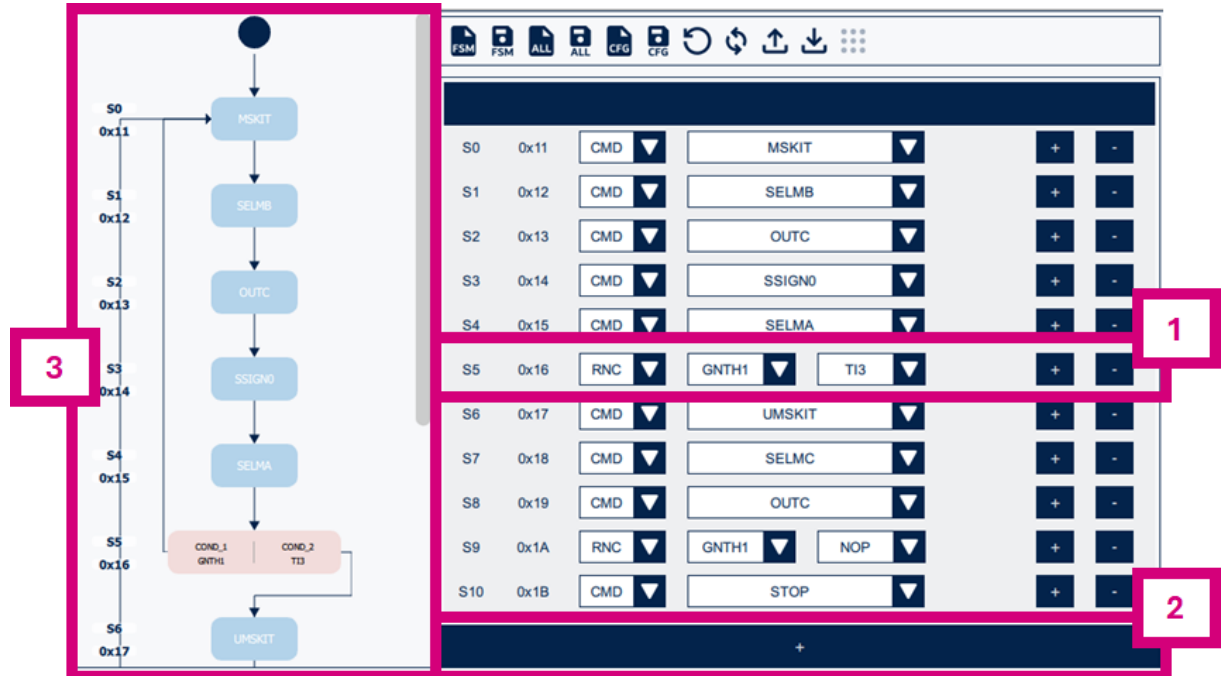
Variable Data		
0x06	Thr... (0.300)	34CD
0x08	Ma...	A8
0x09	Te...	00
0x0A	Ma...	00
0x0B	Te...	00
0x0C	Ma...	01

The **[Variable Data]** group box simplifies the resource allocation process: all the needed resources are automatically shown or hidden in the **[Variable Data]** group box depending on the instructions that compose the **[Instructions Section]**. The user has just to set the values of the shown resources.

14.1.1.4 Instructions Section

The [Instructions Section] group box is available in the center of the [Configuration] tab.

Figure 26. [Configuration] tab – [Instructions Section]



The [Instructions Section] group box helps the user to build the algorithm logic. The [Variable Data] group box is dynamically updated depending on resources used in the [Instructions Section] group box. In the [Instructions Section] group box, more actions can be taken:

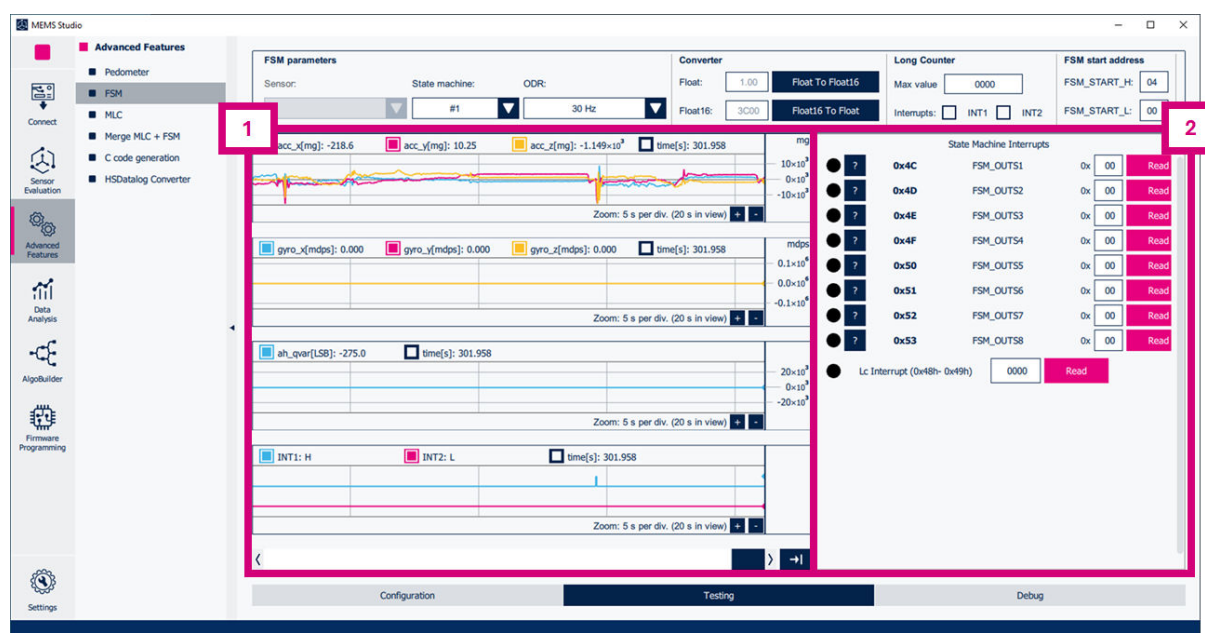
1. Customize an existing state. The single state is composed of:
 - State number Sx
 - State relative hexadecimal address (address 0x00 corresponds to the CONFIG_A byte in the [Fixed Data] section)
 - State type: the user can customize the state type (command or condition) through the CMD / RNC drop-down list as described below:
 - **[RNC]**: the state is a RESET/NEXT condition. In this case, two additional drop-down lists are shown. The left one is related to the RESET condition while the right one is related to the NEXT condition.
 - **[CMD]**: the state is a command. In this case, one additional drop-down list is shown. Commands having one or more parameters (automatically displayed by the tool) require the user to manually configure the parameter values.
 - [+] button is used to insert a new state just before the current one.
 - [-] button is used to remove the current state.
2. Add a new state at the end of the state machine. This button is always positioned at the bottom of the state machine states.
3. Analyze the program flow thanks to the graphical representation of the program.

14.1.2 Testing tab

The **[Testing]** tab of the **[Finite State Machine]** tool allows the user to check the functionality of the configured programs at runtime. The UI is composed of two parts as shown in Figure 27.

- Signal plots: depending on the connected device, the UI shows plots of the data of the enabled sensors and of the enabled interrupt signals.
- [State Machine Interrupts]** status: in this group box, two columns of information are shown:
 - A graphic LED is linked to the corresponding state machine interrupt source bit. By default, the LED is off. When the corresponding source bit is set to 1, the LED is turned on for a while.
 - The FSM_OUTSx register value and the long counter register value can be manually read by clicking on the corresponding **[Read]** button.

Figure 27. [Finite State Machine] tool - [Testing] tab

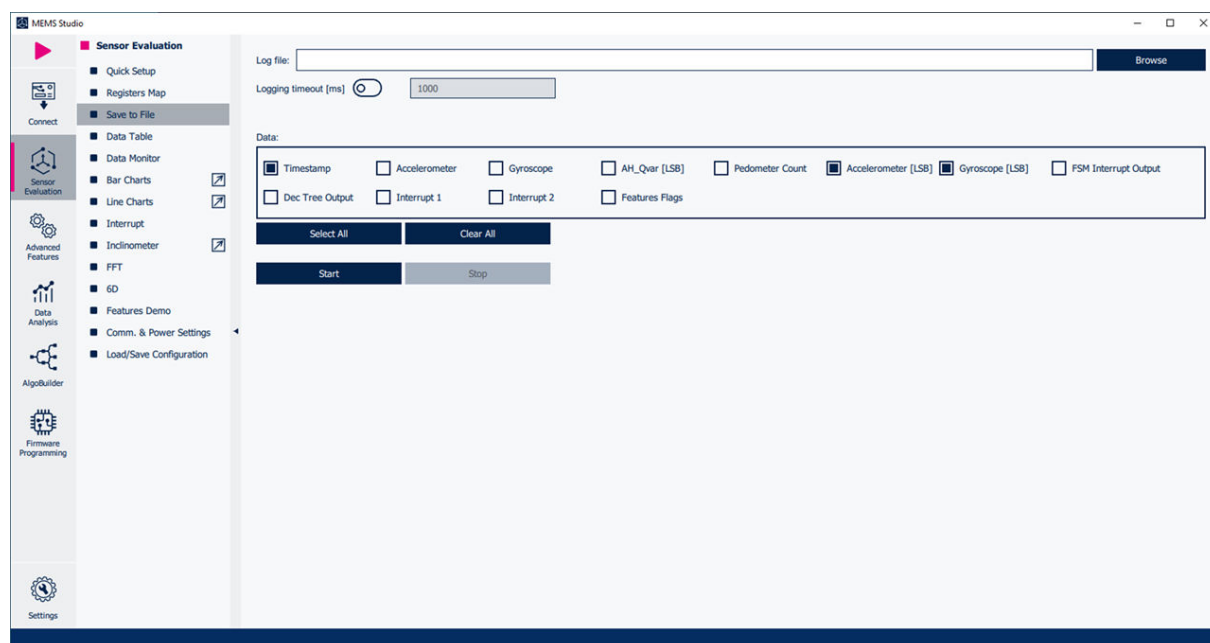


14.1.3 Debug tab

The **[Debug]** tab can be used to inject data into the device in order to check the functionality of the configured programs.

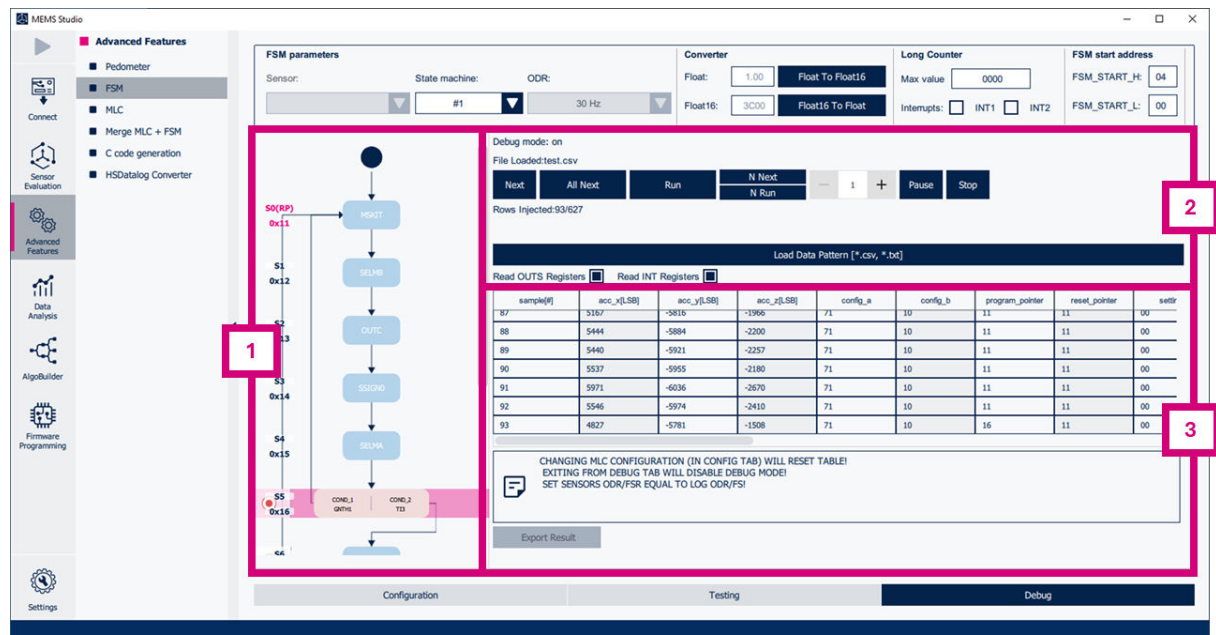
The MEMS Studio **[Save to File]** tab, shown in the following figure, allows the user to collect log files properly formatted for the data injection procedure: if these log files contain [LSB] data, such data will be used during the data injection procedure, otherwise the [LSB] data are calculated by MEMS Studio based on the full-scale settings.

Figure 28. MEMS Studio – [Save to File] tab



The [Debug] tab window is shown in the following figure.

Figure 29. [Finite State Machine] tool – [Debug] tab



The [Debug] tab is mainly composed of three UI parts:

1. State machine flows: the state machine is graphically shown here. When the debug mode is enabled, the current state is highlighted and it is dynamically updated based on the injected sample and program behavior. The UI allows the user to set one or more breakpoints directly on the chart to stop the injection process in case the program pointer reaches one of those states.
2. Debug commands: by default, the debug mode is off. When a log file is loaded, the debug mode is automatically turned on, and the user can start to inject data into the device in order to verify the program functionalities.
3. Output results: after injecting a sample into the device, a new line is added to the table. Table columns represent the state machine parameters and resources, while table rows are related to the injected sample. Finally, it is possible to export the table results in a text file format.

Revision history

Table 12. Document revision history

Date	Version	Changes
09-Apr-2025	1	Initial release

Contents

1	Finite state machine (FSM)	2
1.1	Finite state machine definition	2
1.2	Finite state machine in the LSM6DSV320X	3
2	Signal conditioning block	4
3	FSM block	6
3.1	Configuration block	7
3.1.1	Registers	8
3.1.2	Embedded functions registers	9
3.1.3	Embedded advanced features pages	10
3.2	Program block	12
3.2.1	Input selector block	12
3.2.2	Code block	15
4	FSM interrupt status and signal	17
5	Long counter	18
6	Peak tracking	19
7	FIFO	20
7.1	FSM events in FIFO	20
7.2	Peak-tracking outputs in FIFO	21
8	Fixed Data Section	22
9	Variable Data Section	23
9.1	Thresholds	24
9.2	Extended sinmux	24
9.3	Masks / temporary masks	25
9.4	Angle calculation	26
9.5	TC and timers	26
9.6	Decimator	27
9.7	Previous axis sign	28
9.8	MLC interface	28
10	Instructions Section	29
10.1	RESET/NEXT conditions	29
10.1.1	NOP (0h)	32
10.1.2	TI1 (1h)	32
10.1.3	TI2 (2h)	32
10.1.4	TI3 (3h)	32

10.1.5	TI4 (4h)	33
10.1.6	GNTH1 (5h)	33
10.1.7	GNTH2 (6h)	33
10.1.8	LNTH1 (7h)	33
10.1.9	LNTH2 (8h)	34
10.1.10	GLTH1 (9h)	34
10.1.11	LLTH1 (Ah)	34
10.1.12	GRTH1 (Bh)	35
10.1.13	LRTH1 (Ch)	35
10.1.14	PZC (Dh)	35
10.1.15	NZC (Eh)	35
10.1.16	CHKDT (Fh)	36
10.2	Commands	37
10.2.1	STOP (00h)	38
10.2.2	CONT (11h)	38
10.2.3	CONTREL (22h)	39
10.2.4	SRP (33h)	39
10.2.5	CRP (44h)	39
10.2.6	SETP (55h)	39
10.2.7	SETR (B5h)	40
10.2.8	SETM (F9h)	41
10.2.9	SELMA (66h)	42
10.2.10	SELMB (77h)	42
10.2.11	SELMC (88h)	42
10.2.12	OUTC (99h)	42
10.2.13	STHR1 (AAh)	42
10.2.14	STHR2 (BBh)	42
10.2.15	SELTHR1 (CCh)	43
10.2.16	SELTHR3 (DDh)	43
10.2.17	REL (FFh)	43
10.2.18	SSIGN0 (12h)	43
10.2.19	SSIGN1 (13h)	43
10.2.20	SRTAM0 (14h)	44
10.2.21	SRTAM1 (21h)	44
10.2.22	SINMUX (23h)	45
10.2.23	STIMER3 (24h)	46
10.2.24	STIMER4 (31h)	46
10.2.25	INCR (34h)	47

10.2.26	DECR (FDh)	47
10.2.27	RSTLC (F6h)	47
10.2.28	THRXYZ1 (F7h)	48
10.2.29	THRXYZ0 (F8h)	48
10.2.30	JMP (41h)	49
10.2.31	CANGLE (42h)	49
10.2.32	SMA (43h)	49
10.2.33	SMB (DFh)	49
10.2.34	SMC (FEh)	50
10.2.35	SCTC0 (5Bh)	50
10.2.36	SCTC1 (7Ch)	50
10.2.37	UMSKIT (C7h)	50
10.2.38	MSKITEQ (EFh)	50
10.2.39	MSKIT (F5h)	50
11	FSM configuration example	51
12	Start routine	54
13	Examples of state machine configurations	55
13.1	Toggle	55
13.2	Adaptive self-configuration (ASC)	56
13.3	Free-fall	58
13.4	Decision tree interface	59
13.5	Peak-tracking and extended sinmux	60
14	Finite state machine tool	62
14.1	MEMS Studio	62
14.1.1	Configuration tab	63
14.1.2	Testing tab	66
14.1.3	Debug tab	67
	Revision history	69
	List of tables	73
	List of figures	74

List of tables

Table 1.	Registers	8
Table 2.	Embedded functions registers	9
Table 3.	Embedded advanced features registers - page 0	10
Table 4.	Embedded advanced features registers - page 1	11
Table 5.	Embedded advanced features registers - page 2	11
Table 6.	Finite state machine events in FIFO	20
Table 7.	Format of peak-tracking output data	21
Table 8.	Conditions	30
Table 9.	List of commands.	37
Table 10.	ASC FSM main page registers.	40
Table 11.	ASC FSM embedded functions registers.	40
Table 12.	Document revision history	69

List of figures

Figure 1.	Generic state machine.	2
Figure 2.	State machine in the LSM6DSV320X	3
Figure 3.	Signal conditioning block	4
Figure 4.	FSM block	6
Figure 5.	Program block	12
Figure 6.	FSM inputs (low- <i>g</i> accelerometer).	13
Figure 7.	FSM inputs (high- <i>g</i> accelerometer)	13
Figure 8.	FSM inputs (gyroscope).	14
Figure 9.	FSM program _x code structure.	15
Figure 10.	FSM program _x memory area	16
Figure 11.	[Fixed Data Section]	22
Figure 12.	[Variable Data Section]	23
Figure 13.	Single state description	29
Figure 14.	MLC identifiers for filters and features	46
Figure 15.	FSM configuration example	51
Figure 16.	Toggle state machine example	55
Figure 17.	ASC state machine example	56
Figure 18.	Free-fall state machine example	58
Figure 19.	Decision tree interface example	59
Figure 20.	Peak-tracking example	60
Figure 21.	Running the finite state machine tool	62
Figure 22.	[Finite State Machine] tool - [Configuration] tab	63
Figure 23.	[Configuration] tab - [Status Data]	64
Figure 24.	[Configuration] tab - [Fixed Data]	64
Figure 25.	[Configuration] tab - [Variable Data]	64
Figure 26.	[Configuration] tab - [Instructions Section]	65
Figure 27.	[Finite State Machine] tool - [Testing] tab	66
Figure 28.	MEMS Studio - [Save to File] tab	67
Figure 29.	[Finite State Machine] tool - [Debug] tab	68

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved