life.augmented

AN6207

Application note

ST1VAFE3BX: finite state machine

Introduction

The ST1VAFE3BX is a biosensor embedding a vertical analog front-end (vAFE) channel to detect biopotential signals and a high-performance 3-axis digital accelerometer for motion tracking. The device also embeds an analog hub (AH) sensing functionality that is able to connect an analog input and convert it to a digital signal for embedded processing.

This document provides information on the finite state machine feature available in the ST1VAFE3BX. 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 finite state machine processing capability is obtained through logic connections. A state machine is a mathematical abstraction composed of a finite number of states and transitions between them. The states definition and how they are connected implement the desired logic for pattern recognition.

The ST1VAFE3BX 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.

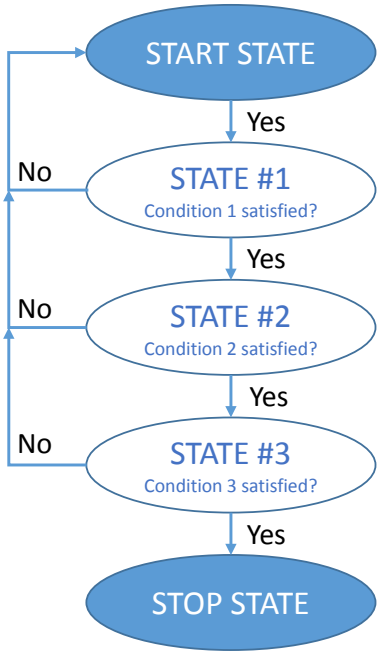
The results of the finite state machine can be read from the application processor at any time. Furthermore, there is the possibility to generate an interrupt when the desired pattern is detected.

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 ST1VAFE3BX

The ST1VAFE3BX is a biosensor embedding a vAFE channel to detect biopotential signals and a 3-axis digital accelerometer. It also embeds an analog hub (AH) sensing functionality that is able to connect an analog input and convert it to a digital signal for embedded processing. These data can be used as the input of up to eight programs in the embedded finite state machine (refer to the following figure).

Figure 2. State machine in the ST1VAFE3BX

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

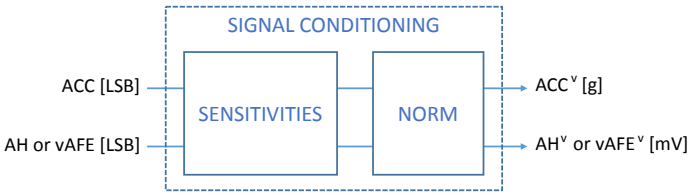
All eight finite state machines are independent: each one has its dedicated memory area and it is independently executed. An interrupt is generated when the end state is reached or when some specific command is performed. Typically, the interrupt is generated when a specific pattern is recognized.

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]) to the following unit conventions by default:

- Accelerometer data in [*g*]
- AH or vAFE data in [mV]

Figure 3. Signal conditioning block



This block applies the sensitivity to [LSB] input data, and then converts these data to half-precision floating-point (HFP) format before passing them to the FSM block. Both the accelerometer and AH / vAFE data conversion factor are automatically handled by the device. Internally, the data are first converted to HFP format, then the proper sensitivity is applied. The AH / vAFE channel is managed as a 3-axis sensor providing only one value of data as X-axis data.

The gain of the AH / vAFE channel is equal to 1311 LSB/mV, corresponding to a sensitivity equal to ~0.780 µV/LSB. The half-precison floating-point representation of this value (1266h) is written in the AH_BIO_SENSITIVITY_L (B6h) and AH_BIO_SENSITIVITY_H (B7h) embedded advanced features registers as the default sensitivity of the AH / vAFE channel. It is possible to modify the applied sensitivity by writing the desired value in these registers. This can be useful when an external sensor is connected to the analog hub interface.

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

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

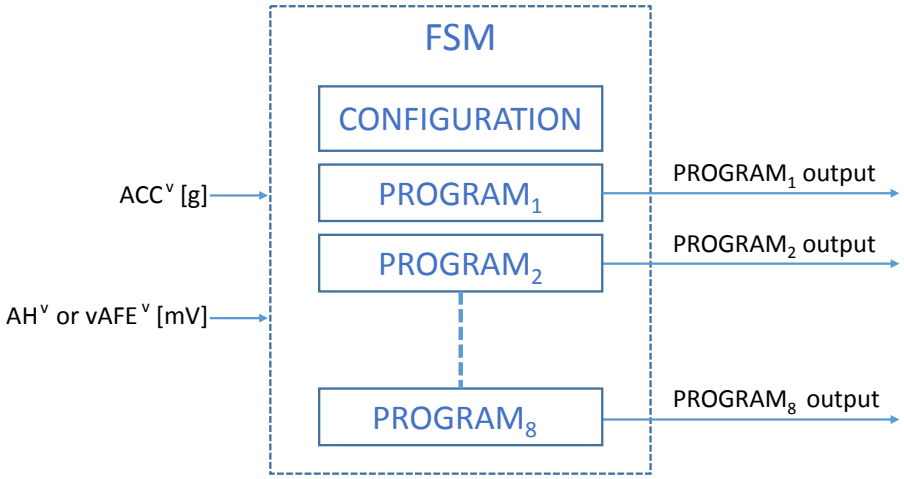
The norm of the input data can be used in the state machine programs, in order to guarantee a high level of program customization for the user.

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, configuration of the programs, and so forth).

The embedded function registers can be used to properly configure the FSM: these registers are accessible when the FUNC_CFG_EN bit of the FUNC_CFG_ACCESS (3Fh) register is set to 1.

The ST1VAFE3BX device is provided with an extended number of registers inside the embedded function 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 3Fh // Enable embedded function 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 3Fh // Disable embedded function 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 (58h) and FSM_START_ADD_H (59h) 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 02h, address FFh to page 03h, address 00h). The ST1VAFE3BX device provides four pages that can be addressed through the PAGE_SEL (02h) register. To address the last page, PAGE_SEL (02h) has to be set to 31h.
- 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 (39h) embedded function register. If the configured FSM execution rate is greater than the configured device ODR, the FSM programs are actually executed at the rate of the accelerometer ODR. This implies that if the device is configured in power-down mode, the FSM programs are not executed.

The FSM_ODR_[2:0] bit mapping changes whether the device is set in the AH / vAFE only state or not. If the device is not set in the AH / vAFE only state, the finite state machine ODR can be configured from 12.5 Hz to 800 Hz. If the device is set in the AH / vAFE only state, the finite state machine ODR can be configured from 50 Hz to 1600 Hz.

For a detailed example of how to configure the entire FSM, refer to Section 9: 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
MD1_CFG	1Fh	-	0	-	-	-	-	-	INT_EMB_FUNC
AH_BIO_CFG1	30h	0	0	0	AH_BIO_ZIN_ DIS_AH2_BIO2	AH_BIO_ZIN_ DIS_AH1_BIO1	0	0	0
AH_BIO_CFG2	31h	0	AH_BIO_MODE1	AH_BIO_MODE0	AH_BIO_ C_ZIN_1	AH_BIO_ C_ZIN_0	AH_BIO_ GAIN_1	AH_BIO_ GAIN_0	AH_BIO_EN
AH_BIO_CFG3	32h	0	0	-	-	0	0	0	AH_BIO_ACTIVE
EMB_FUNC_STATUS_ MAINPAGE	34h	IS_FSM_LC	0	-	-	-	0	0	0
FSM_STATUS_ MAINPAGE	35h	IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1
FUNC_CFG_ACCESS	3Fh	EMB_FUNC_ REG_ACCESS	0	0	0	0	0	0	FSM_WR_ CTRL_EN

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 (3Fh) 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	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_INT	0Ah	INT_FSM_LC	0	-	-	-	0	0	0
FSM_INT	0Bh	INT_FSM8	INT_FSM7	INT_FSM6	INT_FSM5	INT_FSM4	INT_FSM3	INT_FSM2	INT_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	18h	0	0	0	0	FSM_FIFO_EN	-	-	-
FSM_ENABLE	1Ah	FSM8_EN	FSM7_EN	FSM6_EN	FSM5_EN	FSM4_EN	FSM3_EN	FSM2_EN	FSM1_EN
FSM_LONG_COUNTER_L	1Ch	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	1Dh	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	20h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS2	21h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS3	22h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS4	23h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS5	24h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS6	25h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS7	26h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
FSM_OUTS8	27h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V
EMB_FUNC_INIT_B	2Dh	0	0	0	-	0	0	0	FSM_INIT
FSM_ODR	39h	0	1	FSM_ODR_2	FSM_ODR_1	FSM_ODR_0	0	0	0

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.

Note: The content of these registers is loaded when the embedded functions are enabled by setting the EMB_FUNC_EN bit to 1 in the CTRL4 (13h) register. The embedded functions must be enabled in order for these registers to become accessible.

Table 3. Embedded advanced features registers - page 0

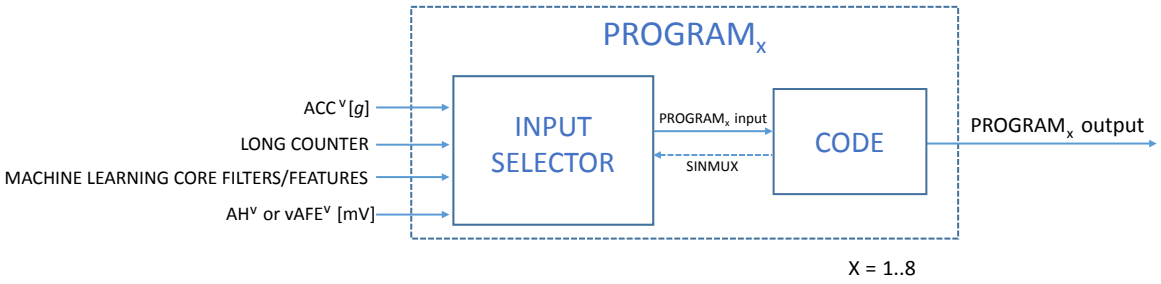
Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC_TIMEOUT_L	54h	FSM_LC_ TIMEOUT7	FSM_LC_ TIMEOUT6	FSM_LC_ TIMEOUT5	FSM_LC_ TIMEOUT4	FSM_LC_ TIMEOUT3	FSM_LC_ TIMEOUT2	FSM_LC_ TIMEOUT1	FSM_LC_ TIMEOUT0
FSM_LC_TIMEOUT_H	55h	FSM_LC_ TIMEOUT15	FSM_LC_ TIMEOUT14	FSM_LC_ TIMEOUT13	FSM_LC_ TIMEOUT12	FSM_LC_ TIMEOUT11	FSM_LC_ TIMEOUT10	FSM_LC_ TIMEOUT9	FSM_LC_ TIMEOUT8
FSM_PROGRAMS	56h	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	58h	FSM_START7	FSM_START6	FSM_START5	FSM_START4	FSM_START3	FSM_START2	FSM_START1	FSM_START0
FSM_START_ADD_H	59h	FSM_START15	FSM_START714	FSM_START13	FSM_START12	FSM_START11	FSM_START10	FSM_START9	FSM_START8
AH_BIO_SENSITIVITY_L	B6h	AH_BIO_S_7	AH_BIO_S_6	AH_BIO_S_5	AH_BIO_S_4	AH_BIO_S_3	AH_BIO_S_2	AH_BIO_S_1	AH_BIO_S_0
AH_BIO_SENSITIVITY_H	B7h	AH_BIO_S_15	AH_BIO_S_14	AH_BIO_S_13	AH_BIO_S_12	AH_BIO_S_11	AH_BIO_S_10	AH_BIO_S_9	AH_BIO_S_8

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:

- ST1VAFE3BX accelerometer data, with precomputed norm (V)
- ST1VAFE3BX AH or vAFE data
- Long counter value
- Internally filtered data and computed features, by properly configuring the machine learning core

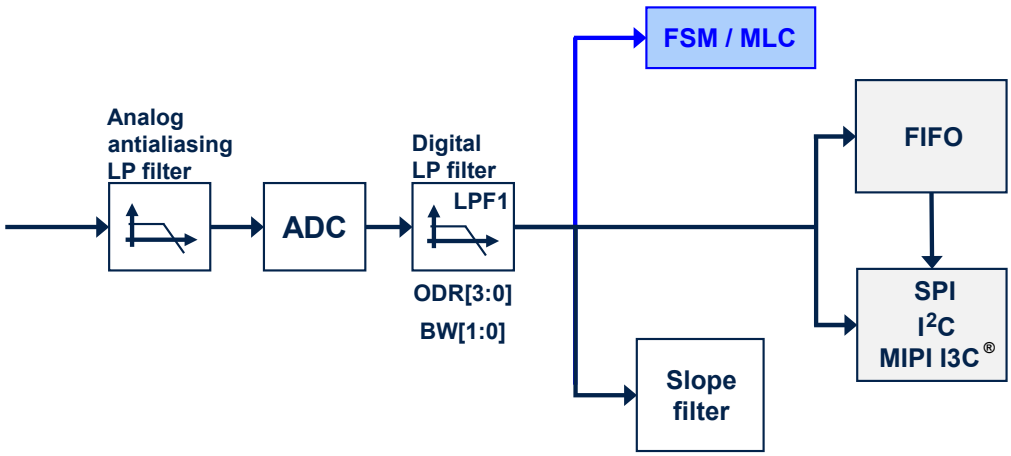
The norm (V) is internally computed with the following formula:

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

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 AN6208.

The following figure shows the inputs of the finite state machine block in the accelerometer digital chain.

Figure 6. FSM inputs (accelerometer)



The signal bandwidth of the accelerometer depends on the device configuration. For additional information, refer to AN6160 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.

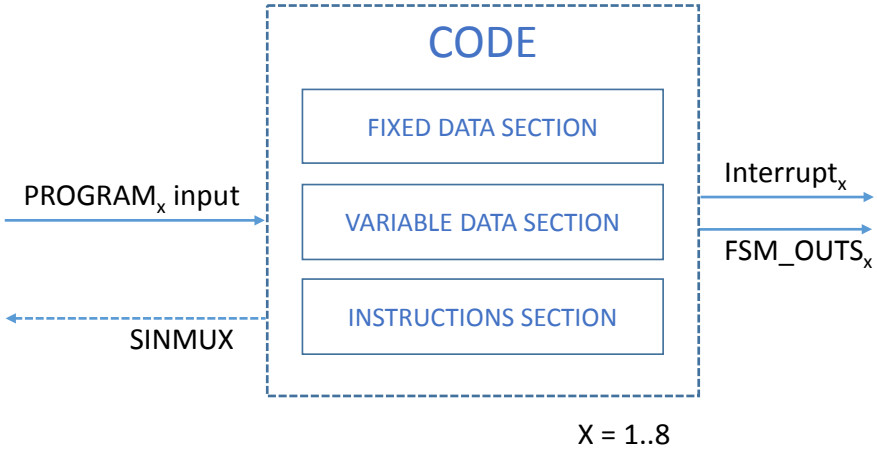
3.2.2 Code block

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 input$_x$ signal.

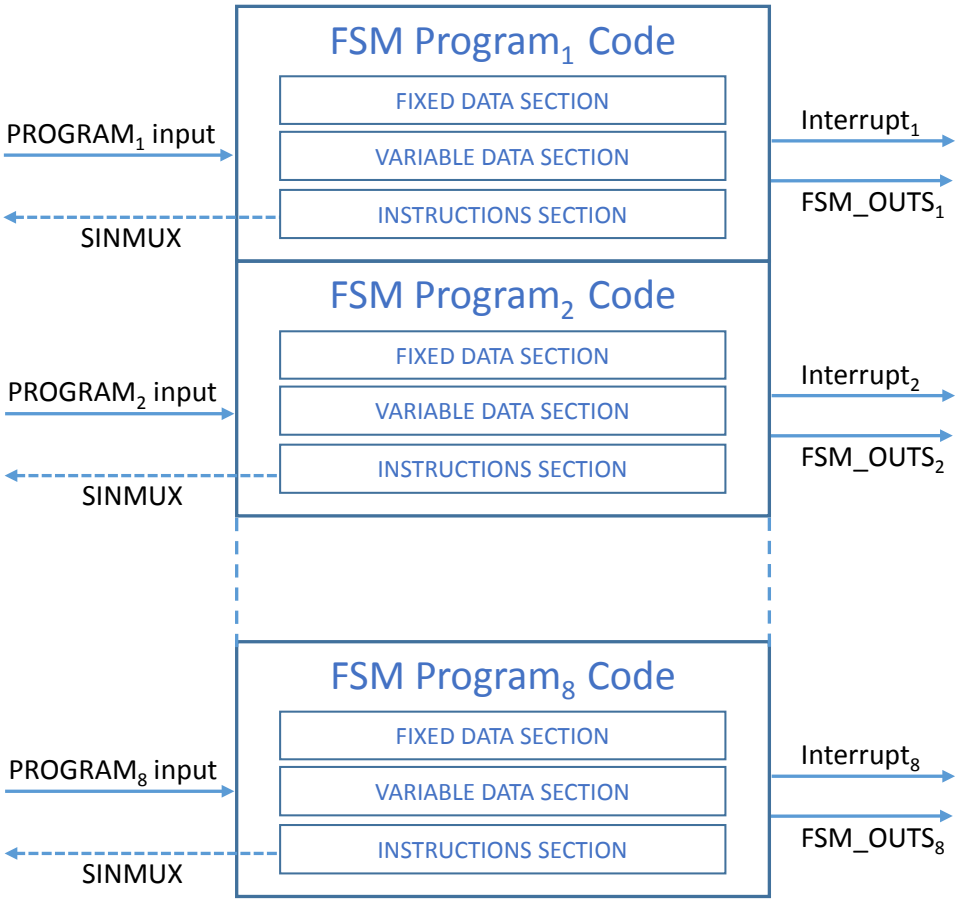
Figure 7. FSM program$_x$ code structure



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 or each time the EMB_FUNC_EN bit of the CTRL4 (13h) register is reset since the embedded functions domain is powered off and the embedded registers content is lost.

Figure 8. **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 function register.

The FSM long counter interrupt signal is generated when the long counter value, stored in the FSM_LONG_COUNTER_L (1Ch) and FSM_LONG_COUNTER_H (1Dh) embedded function registers, reaches the configured long counter timeout value stored in the FSM_LC_TIMEOUT_L (54h) and FSM_LC_TIMEOUT_H (55h) embedded advanced features registers (page 0).

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

- FSM_STATUS_MAINPAGE (35h) register or FSM_STATUS (13h) embedded function register for the FSM interrupt status
- EMB_FUNC_STATUS_MAINPAGE (34h) register or EMB_FUNC_STATUS (12h) embedded function register for the long counter interrupt status

The FSM interrupt signal can be driven to the INT interrupt pin by setting the dedicated bit:

- INT_FSM[1:8] bit of the FSM_INT embedded function register to 1
- INT_FSM_LC bit of the EMB_FUNC_INT embedded function register to 1

Note: In both of the above cases it is mandatory to also enable routing the embedded functions events to the INT interrupt pin by setting the INT_EMB_FUNC bit of the MD1_CFG register.

The behavior of the interrupt signal is pulsed by default. The duration of the pulse is 1/ODR.

Note: The minimum pulse duration is 1/1600 Hz (~625 µs).

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 (35h) register or FSM_STATUS (13h) embedded function register for the FSM interrupt
- EMB_FUNC_STATUS_MAINPAGE (34h) register or EMB_FUNC_STATUS (12h) embedded function register for the long counter interrupt

5 Long counter

The long counter is a 15-bit temporary counter resource available to the user. It is possible to increment, decrement, or reset its value, stored in the FSM_LONG_COUNTER_L (1Ch) and FSM_LONG_COUNTER_H (1Dh) registers, by using the INCR, DECR, or RESET command, respectively. 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 (54h) and FSM_LC_TIMEOUT_H (55h) 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 (34h) register and EMB_FUNC_STATUS (12h) embedded function 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**].

If any of the programs makes use of the long counter through the commands described above, the FIFO must be enabled in continuous mode. Refer to application note AN6160 for more details on the FIFO buffer.

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 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 9. [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	0	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: Green colored bits have to be set according to program purposes, while red bits have to be set to 0 when the program is loaded into the embedded advanced features registers page (they are automatically configured by the FSM logic).

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 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 10: 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.

7 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 proper size and at the proper position. The following figure shows the structure of the [**Variable Data Section**].

Figure 10. [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	TC	TC(15:0) or TC(7:0)							
21									
22	TIMER1	TIMER1(15:0)							
23									
24	TIMER2	TIMER2(15:0)							
25									
26	TIMER3	TIMER3(7:0)							
27	TIMER4	TIMER4(7:0)							
28	DEST	DEST(7:0)							
29	DESC	DESC(7:0)							
30	PAS	SCTC	0	MSKIT	MSKITEQ	SIGN_X	SIGN_Y	SIGN_Z	SIGN_V
31	DECTREE	0	0	DTSEL(1:0)		DTRES(3:0)			

As shown in the table above, the maximum size of the [**Variable Data Section**] is 26 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).*

7.1 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 ST1VAFE3BX accelerometer signal is selected, the unit of the threshold is [*g*].
- If the ST1VAFE3BX AH or vAFE signal is selected and the sensitivity written in the AH_BIO_SENSITIVITY_L (B6h) and AH_BIO_SENSITIVITY_L (B7h) embedded advanced features registers is configured at the default value, the unit is [mV]. Otherwise, if the sensitivity is configured at a different value, the unit depends on the applied conversion factor.
- 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

7.2 Extended sinmux

The extended sinmux resource is mainly used to select as FSM inputs the long counter value (SINMUX with argument 8) or a feature / filter computed by the machine learning core (SINMUX with first argument 9). 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 8 (long counter selection) or 9 (MLC feature or filter selection)
- THRXYZ0, THRXYZ1

Involved conditions:

- N/A

7.3 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. If the input data is generated by the ST1VAFE3BX AH / vAFE interface, 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

7.4 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 (39h) 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, theTC resource occupies two bytes.

Involved commands:

- STIMER3 / STIMER4
- SCTC0 / SCTC1

Involved conditions:

- TI1 / TI2 / TI3 / TI4

7.5 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 (39h) register, according to the following formula:

PROGRAM_ODR = FSM_ODR / DEST

At startup:

DESC = DEST (initial decimation value)

When the sample clock occurs:

DESC = 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: The minimum meaningful value for DEST is '2'.

7.6 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) 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 / MSKIT / MSKITEQ / UMSKIT

Involved conditions:

- PZC / NZC

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

7.7 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 7.2: Extended sinmux and Section 8.2.21: 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 application note AN6208 for more details about how to configure the MLC.

Involved commands:

- SINMUX

Involved conditions:

- CHKDT

8 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.

8.1 RESET/NEXT conditions

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

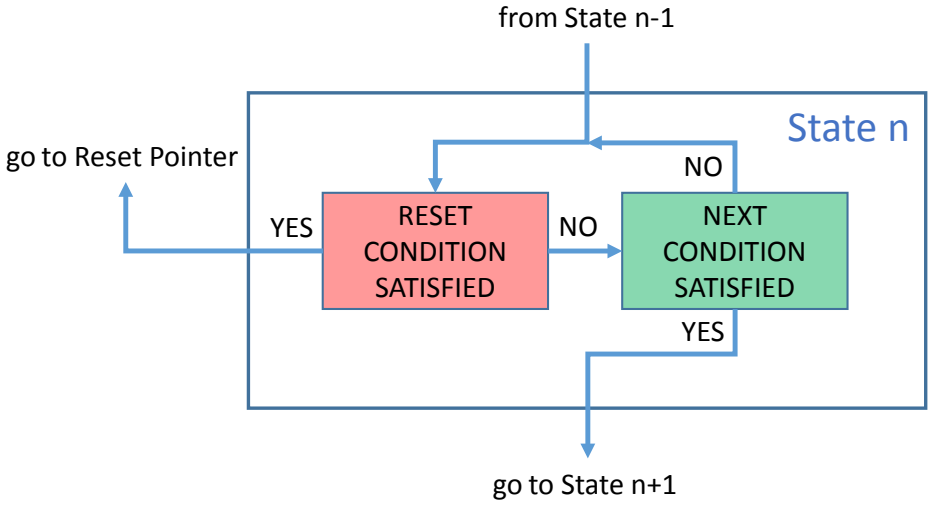
The RESET condition is defined in the opcode MSB part while the NEXT condition is defined in the opcode LSB part.

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 11. 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 8.2.27: THRXYZ1 (F7h) and Section 8.2.28: 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 7.7: MLC interface for more details).

Table 4. 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 ≥ THRESH1	Input signal, triggered with mask, compared to threshold	THRESH1, one MASK
6h	GNTH2	Any triggered axis ≥ 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 ≥ THRESH1		THRESH1, one MASK
Ah	LLTH1	All triggered axes < THRESH1		THRESH1, one MASK
Bh	GRTH1	Any triggered axis ≥ -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 | PZC opcode (0xFD) is equal to DECR opcode.*
- *CHKDT | NZC opcode (0xFE) is equal to SMC opcode.*

8.1.1 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.

8.1.2 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 = 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 8.2.27: THRXYZ1 (F7h) and Section 8.2.28: THRXYZ0 (F8h).

8.1.3 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 = 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 8.2.27: THRXYZ1 (F7h) and Section 8.2.28: THRXYZ0 (F8h).

8.1.4 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 = 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 8.2.27: THRXYZ1 (F7h) and Section 8.2.28: THRXYZ0 (F8h).

8.1.5 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 = 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 8.2.27: THRXYZ1 (F7h) and Section 8.2.28: THRXYZ0 (F8h).

8.1.6 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 ≥ threshold || -X ≥ 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.

8.1.7 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 ≥ threshold || -X ≥ 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.

8.1.8 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 < threshold || -X < 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.

8.1.9 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 < threshold || -X < 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.

8.1.10 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 ≥ threshold || -X ≥ 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.

8.1.11 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 < threshold || -X < 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.

8.1.12 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 ≥ threshold || -X ≥ 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.

8.1.13 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 < threshold || -X < 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.

8.1.14 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.

8.1.15 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.

8.1.16 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 7.7: 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.

8.2 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 that 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 5. 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)
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 8.2.21: SINMUX (23h) for details about the parameters that are needed.
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

Opcode	Mnemonic	Description	Parameter
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
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

8.2.1 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 (1Ah) register to restart the program. In this case, the start routine is performed. For additional information about the start routine, refer to Section 10: Start routine.

8.2.2 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

8.2.3 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

8.2.4 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

8.2.5 CRP (44h)

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

Parameters: none

Actions:

- RP = beginning of program code
- PP = PP + 1

8.2.6 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

- 1^{st} 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).
- 2^{nd} parameter: new value (8 bits) to be written in the 1^{st} parameter address

Actions:

- Byte value addressed by 1^{st} parameter = 2^{nd} parameter
- PP = PP + 3

8.2.7 SETR (B5h)

8.2.7.1 ASC feature

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

Table 6. ASC FSM main page registers

Main page register	Register address (master)	Register address (FSM)
CTRL1	10h	10h
CTRL2	11h	11h
CTRL3	12h	12h
CTRL4	13h	13h
CTRL5	14h	14h
FIFO_CTRL	15h	15h(1)

1. *FIFO_MODE change not supported.*

Table 7. ASC FSM embedded functions registers

Embedded functions register	Register address (master)	Register address (FSM)
EMB_FUNC_EN_A	04h	01h(1)
EMB_FUNC_EN_B	05h	02h(1)
EMB_FUNC_FIFO_EN	18h	05h(1)
FSM_ENABLE	1Ah	03h(1)

1. *Access to the embedded function 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 (3Fh) register is used to give write capability of the above device registers to either the master or the FSM. After writing this bit, the controller change occurs after 50 µs.

When accessing the embedded functions registers by setting the EMB_FUNC_REG_ACCESS bit of the FUNC_CFG_ACCESS register to 1, the FUNC_CFG_ACCESS register (which also contains the FSM_WR_CTRL_EN bit) cannot be read, but it can be written. In order to set the FSM_WR_CTRL_EN bit to the desired value without running the risk of overwriting its content when entering/exiting the embedded functions registers page, in the application code it is recommended to set it to the desired value before accessing the embedded functions registers, while also storing said value in a byte-sized variable. Then, it is recommended to follow these procedures to enter/exit the embedded functions registers page:

- When entering the embedded functions register page, instead of writing 80h in the FUNC_CFG_ACCESS register, write the bitwise OR operation between 80h and the value stored in the variable in the register.
- When exiting the embedded functions register page, instead of writing 00h in the FUNC_CFG_ACCESS register, write the value stored in the variable in the register.

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 bandwidth and full scale, two consecutive SETR commands must be performed as described below:

1. *SETR 0x00 0xF0 (bitmask equal to 0xF0)*
2. *SETR 0x14 0x80 (new accelerometer ODR equal to 100 Hz)*

8.2.8 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

8.2.9 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

8.2.10 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

8.2.11 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

8.2.12 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

8.2.13 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

8.2.14 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

8.2.15 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

8.2.16 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

8.2.17 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

8.2.18 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

8.2.19 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

8.2.20 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

8.2.21 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 be also 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 9, otherwise one byte.

- 1st parameter: value to select input source:
  - 0: accelerometer [a_x a_y a_z a_v]
  - 1: AH / vAFE [m_x 0 0 0]
  - 3: first filtered signal from the machine learning core(1) [F_x F_y F_z F_v(2)]
  - 4: third filtered signal from the machine learning core(1) [H_x H_y H_z H_v(2)]
  - 5: second filtered signal norm from the machine learning core(1) [0 0 0 G_v(3)]
  - 6: fourth filtered signal norm from the machine learning core(1) [0 0 0 J_v(3)]
  - 8: long counter value [LC_x 0 0 0]
  - 9: 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 9): MLC filter or feature IDENTIFIER[7:0]
- 3rd parameter (needed only if the 1st parameter is equal to 9): MLC filter or 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 12. 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 processing of the AH / vAFE signal is selected by using the SINMUX 1 command. In this case, the mask value to be used during a comparison can be 0x80 (corresponding to +X axis) or 0x40 (corresponding to -X axis).

Note: When a filter on the axes is intended to be selected through the SINMUX 9 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.

8.2.22 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

8.2.23 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

8.2.24 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 (1Ch) and FSM_LONG_COUNTER_H (1Dh) embedded functions registers, and is clamped to the long counter timeout value stored into the FSM_LC_TIMEOUT_L (54h) and FSM_LC_TIMEOUT_H (55h) 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

8.2.25 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 (1Ch) and FSM_LONG_COUNTER_H (1Dh) embedded functions registers, and is clamped to zero.

Parameters: none

Actions:

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

8.2.26 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 (1Ch) and FSM_LONG_COUNTER_H (1Dh) embedded functions registers.

Parameters: none

Actions:

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

8.2.27 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.
- The SINMUX 0 command is not supported and must be replaced with a SINMUX 9 01D4h command, which means setting the IDENTIFIER[7:0] parameter equal to D4h and the IDENTIFIER[15:8] parameter equal to 01h.

Parameters: none

Actions:

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

8.2.28 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

8.2.29 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.

8.2.30 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

8.2.31 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

8.2.32 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

8.2.33 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

8.2.34 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

8.2.35 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

8.2.36 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

8.2.37 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

9 FSM configuration example

This section contains an example that explains all write operations that have to be done in order to configure the ST1VAFE3BX FSM. A few steps have to be followed:

- Configure the FSM registers inside the embedded function registers set.
- Configure the FSM registers inside the embedded advanced features registers set.
- Configure the ST1VAFE3BX accelerometer sensor.

In this example, two simple programs are configured:

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

Both algorithms are intended to use accelerometer data only at a sample rate of 25 Hz.

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

Figure 13. FSM configuration example

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

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

1. Write 00h to register 14h // Set accelerometer sensor in power-down mode
2. Write 10h to register 13h // Enable the embedded functions
3. Wait 5 ms // Wait for the embedded functions to be turned on
4. Write 80h to register 3Fh // Enable access to the embedded function registers
5. Write 00h to register 04h // EMB_FUNC_EN_A = 0

6. Write 00h to register 05h // EMB_FUNC_EN_B = 0
7. Write 48h to register 39h // FSM_ODR[2:0] = 001 (25 Hz)
8. Write 03h to register 1Ah // FSM_ENABLE = 03h
9. Write 00h to register 0Ah // EMB_FUNC_INT = 00h
10. Write 03h to register 0Bh // FSM_INT = 03h
11. Write 40h to register 17h // PAGE_RW: enable write operation
12. Write 01h to register 02h // PAGE_SEL = 0
13. Write 54h to register 08h // PAGE_ADDRESS = 54h
14. Write 00h to register 09h // Write 00h to register FSM_LONG_COUNTER_L
15. Write 00h to register 09h // Write 00h to register FSM_LONG_COUNTER_H
16. Write 02h to register 09h // Write 02h to register FSM_PROGRAMS
17. Write 58h to register 08h // PAGE_ADDRESS = 58h
18. Write 00h to register 09h // Write 00h to register FSM_START_ADDRESS_L
19. Write 02h to register 09h // Write 02h to register FSM_START_ADDRESS_H
20. Write 21h to register 02h // PAGE_SEL = 2
21. Write 00h to register 08h // PAGE_ADDRESS = 00h
22. Write 51h to register 09h // CONFIG_A
23. Write 00h to register 09h // CONFIG_B
24. Write 10h to register 09h // SIZE
25. Write 00h to register 09h // SETTINGS
26. Write 0Ch to register 09h // RESET POINTER
27. Write 00h to register 09h // PROGRAM POINTER
28. Write AEh to register 09h // THRESH1 LSB
29. Write B7h to register 09h // THRESH1 MSB
30. Write 80h to register 09h // MASKA
31. Write 00h to register 09h // TMASKA
32. Write 00h to register 09h // TC
33. Write 10h to register 09h // TIMER3
34. Write 53h to register 09h // GNTH1 | TI3
35. Write 99h to register 09h // OUTC
36. Write 50h to register 09h // GNTH1 | NOP
37. Write 00h to register 09h // STOP (mandatory for having even SIZE bytes)
38. Write 50h to register 09h // CONFIG_A
39. Write 00h to register 09h // CONFIG_B
40. Write 0Ch to register 09h // SIZE
41. Write 00h to register 09h // SETTINGS
42. Write 0Ah to register 09h // RESET POINTER
43. Write 00h to register 09h // PROGRAM POINTER
44. Write 66h to register 09h // THRESH1 LSB
45. Write 3Ch to register 09h // THRESH1 MSB
46. Write 02h to register 09h // MASKA
47. Write 00h to register 09h // TMASKA
48. Write 05h to register 09h // NOP | GNTH1
49. Write 22h to register 09h // CONTREL

50. Write 00h to register 04h // EMB_FUNC_EN_A = 0
51. Write 01h to register 05h // EMB_FUNC_EN_B(FSM_EN) = 1
52. Write 00h to register 17h // PAGE_RW: disable write operation
53. Write 00h to register 3Fh // Disable access to the embedded function registers
54. Write 01h to register 1Fh // MD1_CFG(INT_EMB_FUNC) = 1
55. Write 62h to register 14h // CTRL4 = 62h (25 Hz, 8 *g*)

10 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).

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 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)

11 Examples of state machine configurations

11.1 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 14. 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 performed 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.

11.2 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 ultralow-power mode at 25 Hz. When a wake-up event is detected, the accelerometer is configured in low-power mode at 100 Hz. If stationary for a while, the accelerometer is set back to ultralow-power mode at 25 Hz.

Figure 15. ASC 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	1	0	0	0	0
02h	SIZE	1Ah (26 bytes)							
03h	SETTINGS	00		0	0	0	00		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h 07h	THRESH1	3C66h (1.100)							
08h	MASKA	02h (+V)							
09h	TMASKA	00h							
0Ah	TC	00h							
0Bh	TIMER3	7Dh (125 samples)							
0Ch	PAS	00h							
0Dh	MSKIT	F5h							
0Eh	SETR	B5h							
0Fh	14h	14h							
10h	30h	30h							
11h	NOP | GNTH1	05h							
12h	SETR	B5h							
13h	14h	14h							
14h	80h	80h							
15h	SRP	33h							
16h	GNTH1 | TI3	53h							
17h	CRP	44h							
18h	CONTREL	22h							
19h	STOP	00h							

Instructions section description

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

PP = 0Eh: SETR command is performed without needing a sample set. The register 14h is set to 32h (the accelerometer sensor is configured in ultralow-power mode at 25 Hz). PP = PP + 3.

PP = 11h: 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 = 12h: SETR command is performed without needing a sample set. The register 14h is set to 80h (the accelerometer sensor is configured at 100 Hz). PP = PP + 3.

PP = 15h: SRP command is performed without needing a sample set. The RESET POINTER is set to the next state, 16h. PP = PP + 1.

PP = 16h: 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 = 17h: CRP command is performed without needing a sample set. The RESET POINTER is set to its default value, 0Dh. PP = PP + 1.

PP = 18h: CONTREL command is performed 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 125 samples.

11.3 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 16. 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 07h	THRESH1	34CDh (0.300)							
08h	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 performed 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 performed without the need of 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 consecutives samples, then the PP is increased (PP = PP + 1).

PP = 0Fh: OUTC command is performed 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 25 Hz, the free-fall duration is ~120 ms (three samples at 25 Hz).

11.4 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 to 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 (50 samples having an ODR equal to 25 Hz)

The FSM implements a simple wakeup 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 17. 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	MASKA	02h (+V)							
09h	TMASKA	00h							
0Ah	TC	00h							
0Bh	TIMER3	02h (2 samples)							
0Ch	TIMER4	33h (51 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 performed without the need of a sample set. This generates an interrupt and increases the PP (PP = PP + 1).

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

12 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.

12.1 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

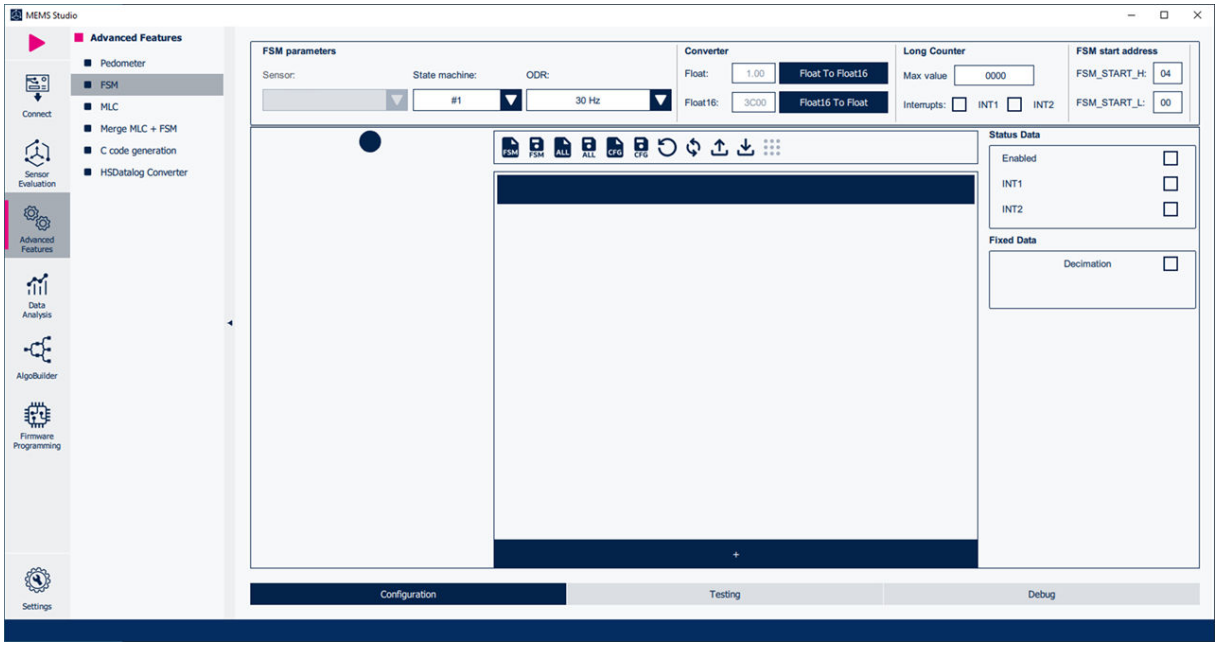
The MEMS Studio GUI allows visualization of sensor outputs in both graphical and numerical format and allows the user to save or generally manage data coming from the device.

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. In this way, the sensor can be reprogrammed in a few 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 tool used for programming the finite state machines, the user first needs to click on the [**Advanced Features**] button that is available in the left side of MEMS Studio and then on the [**FSM**] button. When loaded, the [**Finite State Machine**] tool window is shown as in the following figure.

Figure 18. 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:

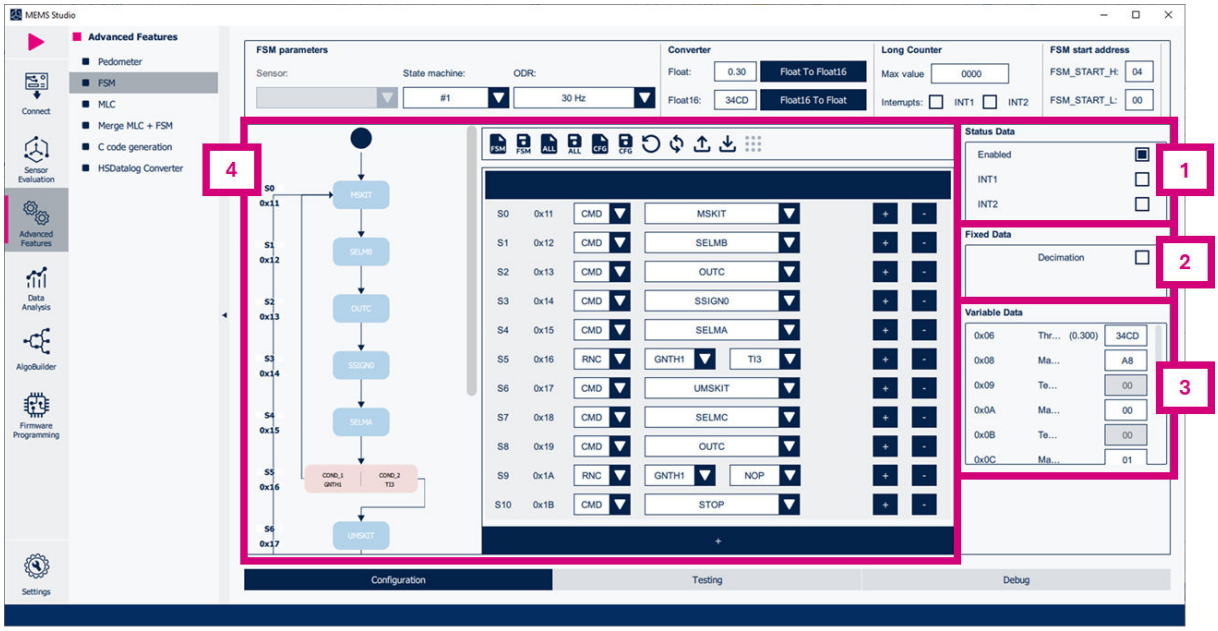
- [**Configuration**] tab (the one selected by default)
- [**Testing**] tab
- [**Debug**] tab

12.1.1 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 19. [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 or .ucf 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, .h or .ucf 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).

12.1.1.1 *Status Data*

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

Figure 20. [Configuration] tab - [Status Data]

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.

12.1.1.2 *Fixed Data*

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

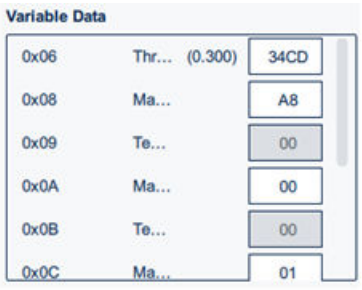
Figure 21. [Configuration] tab - [Fixed Data]

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**].

12.1.1.3 *Variable Data*

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

Figure 22. [Configuration] tab – [Variable Data]

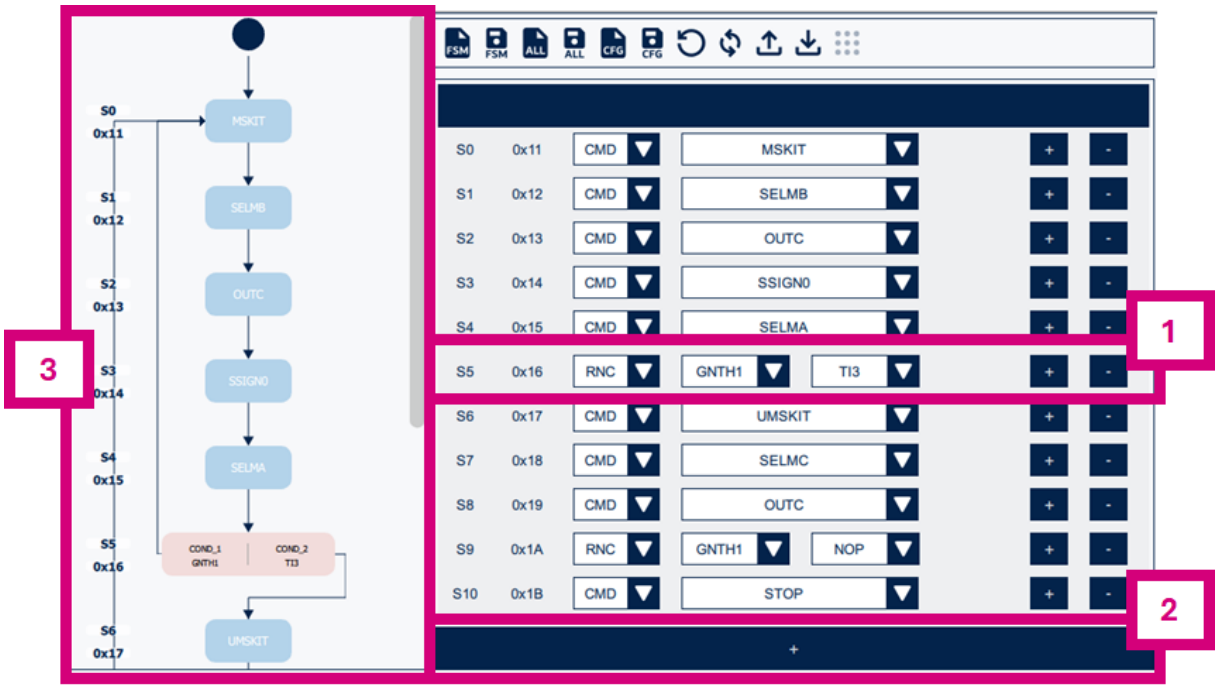


The [**Variable Data**] group box simplifies the resource allocation process: all the needed resources are automatically shown or hidden in the [**Variable Data**] group box depending on the instructions that compose the [**Instructions Section**]. The user has just to set the values of the shown resources.

12.1.1.4 *Instructions Section*

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

Figure 23. [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:

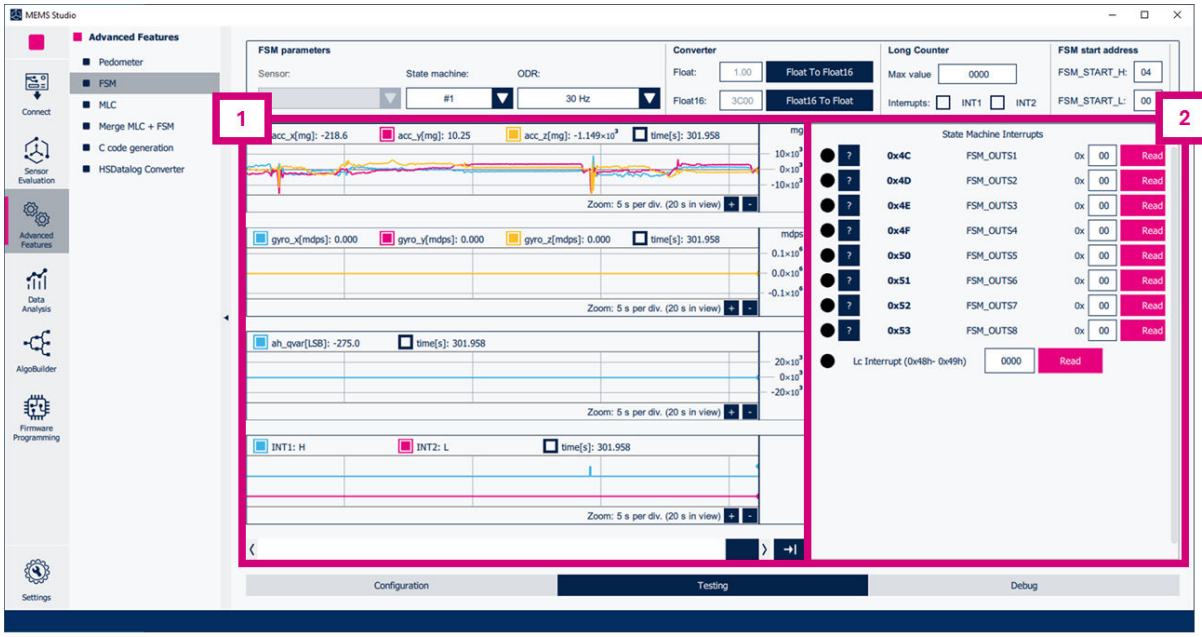
1. Customize an existing state. The single state is composed of:
   - State number Sx
   - State relative hexadecimal address (address 0x00 corresponds to the CONFIG_A byte in the [**Fixed Data**] section)
   - State type: the user can customize the state type (command or condition) through the CMD / 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.

12.1.2 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 24.

1. Signal plots: depending on the connected device, the UI shows plots of the data of the enabled sensors and of the enabled interrupt signals.
2. [**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 24. [Finite State Machine] tool - [Testing] tab

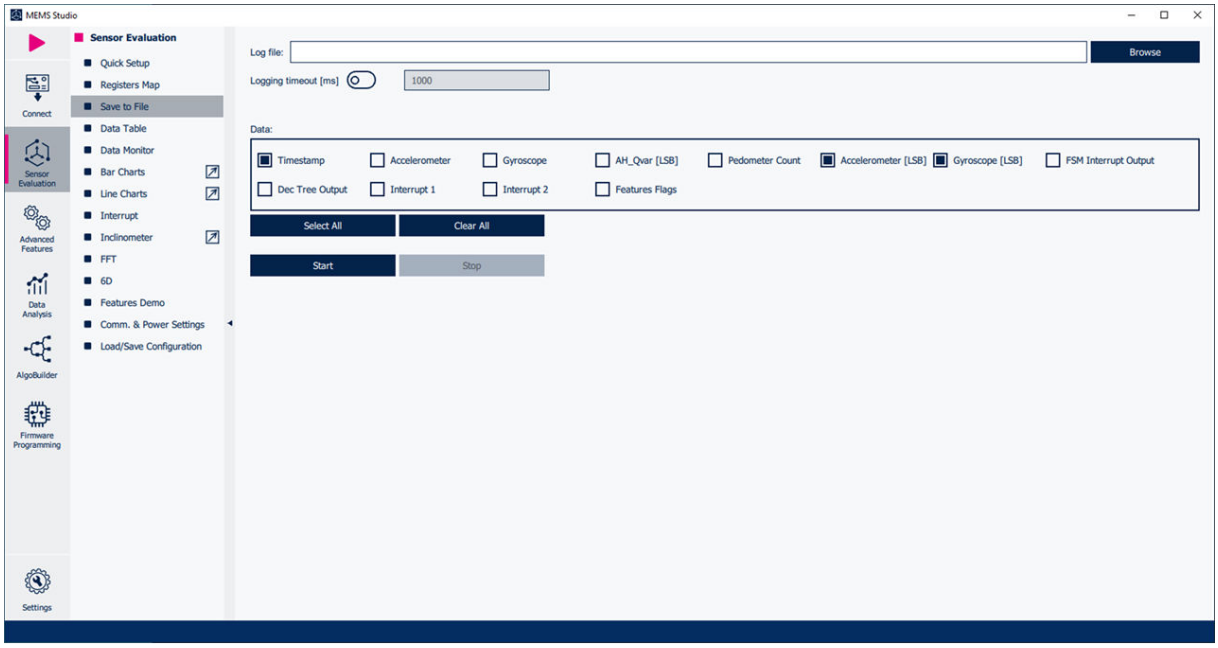


12.1.3 Debug tab

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

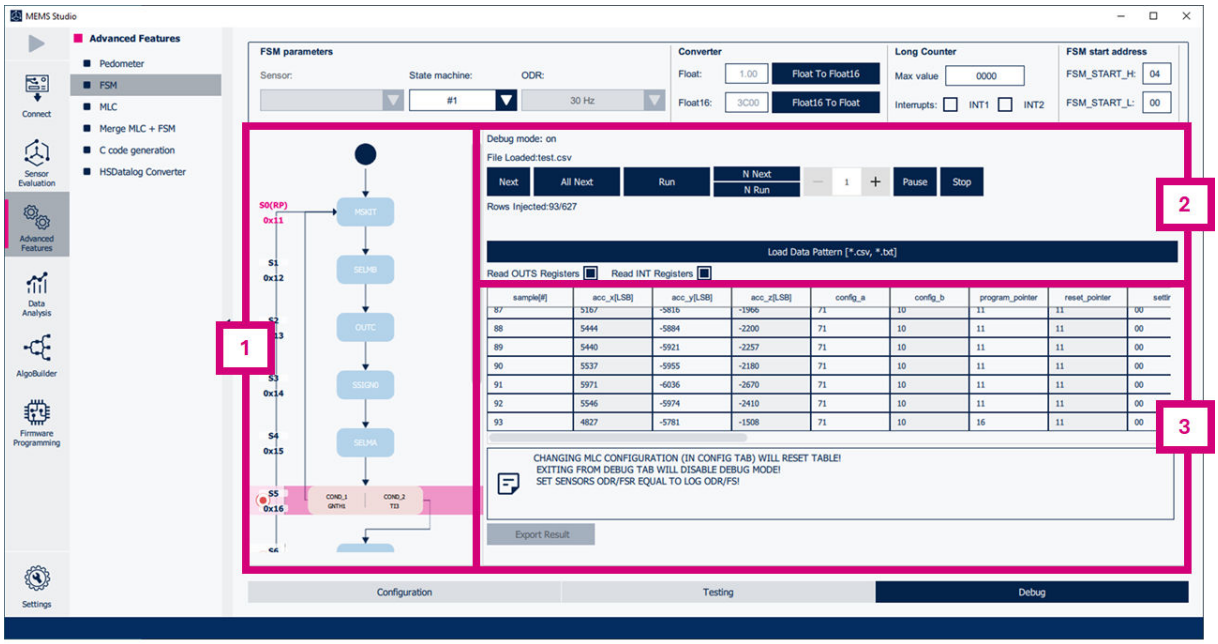
The MEMS Studio [**Save to File**] tab, shown in the following figure, allows the user to collect log files properly formatted for the data injection procedure: these log files must contain [LSB] data only.

Figure 25. MEMS Studio – [Save to File] tab



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

Figure 26. [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 8. **Document revision history**

Date	Version	Changes
25-Oct-2024	1	Initial release

Contents

List of tables

List of figures