life.augmented

AN5542

Application note

IIS2ICLX: Finite State Machine

Introduction

This document is intended to provide information on the use and configuration of ST's IIS2ICLX embedded Finite State Machine.

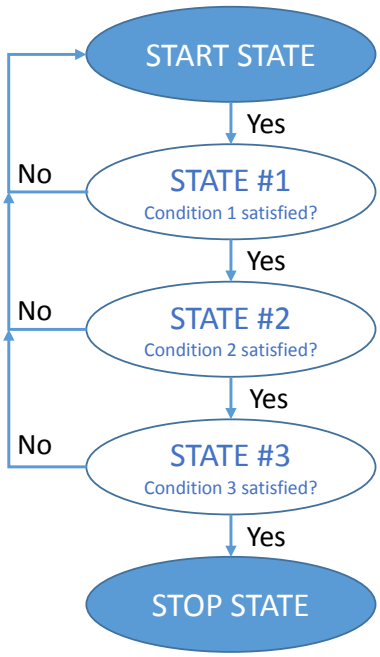
The IIS2ICLX can be configured to generate interrupt signals activated by user-defined motion patterns. For this purpose, up to 16 embedded finite state machines can be programmed independently for motion detection.

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 IIS2ICLX

The IIS2ICLX works as a 2-axis linear accelerometer sensor, generating acceleration output data; it is also possible to connect an external sensor (e.g. gyroscope) by using the sensor hub feature (Mode 2). All these data can be used as input of up to 16 programs in the embedded Finite State Machine (refer to the following figure).

Figure 2. State machine in the IIS2ICLX

The FSM structure is highly modular: it is possible to easily write up to 16 programs, each one able to recognize a specific gesture.

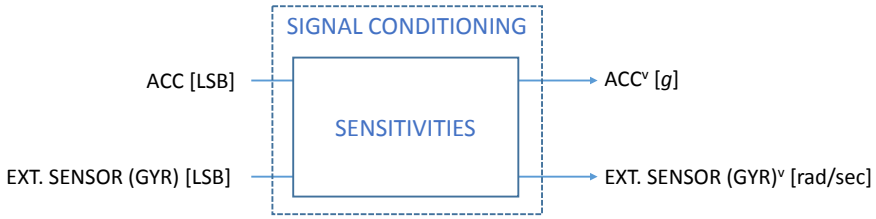
All 16 finite state machines are independent: each one has its dedicated memory area and it is independently executed. An interrupt is generated when the end state is reached or when some specific command is performed. Typically, the interrupt is generated when a specific gesture is recognized.

2 Signal Conditioning block

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

- accelerometer data in [*g*];
- external sensor: if it's a gyroscope, data have to be converted to [rad/sec].

Figure 3. Signal Conditioning block



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

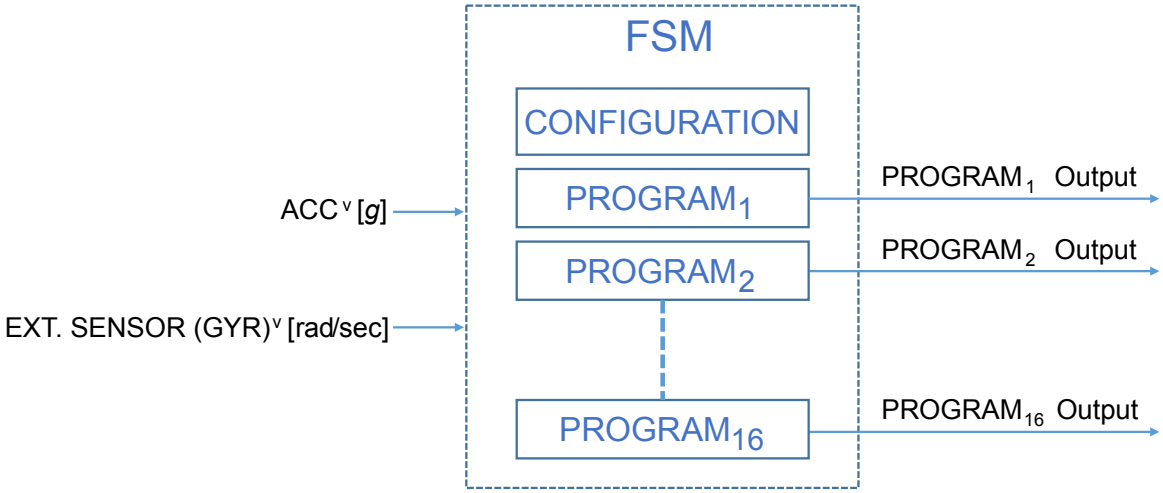
- IIS2ICLX's accelerometer data conversion factor is automatically handled by the device;
- external sensor data conversion factor is not automatically handled by the device: the user has to follow the procedure below in order to set properly the (e.g.) gyroscope conversion factor in the device. Please note that gyroscope data have to be converted to [rad/sec], expressed in HFP format.

3 FSM block

Output data signals coming from the Signal Conditioning block are sent to the FSM block which 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 16 configurable programs: each program processes input data and generates an output.

Figure 4. FSM block



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

3.1 Configuration block

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

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

The IIS2ICLX 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 01h // 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 01h // 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 (7Eh) and FSM_START_ADD_H (7Fh) registers. All programs have to be written in consecutive registers, including two important aspects:

- both the PAGE_SEL (02h) register and PAGE_ADDRESS (08h) register have to be properly updated when moving from one page to another (i.e. when passing from page 03h, address FFh to page 04h, address 00h). The IIS2ICLX device provides 8 pages that can be addressed through the PAGE_SEL (02h) register. To address the last page, PAGE_SEL (02h) has to be set to 71h;
- program SIZE byte must be an even number: if it is odd, an additional STOP state has to be added at the end of the instruction section.

For a detailed example on how to configure the entire FSM, refer to Section 8 FSM configuration example.

3.1.1 FSM registers

The table given below provides a list of the registers related to the FSM and the corresponding addresses.

Table 1. FSM registers

Register name	Type	Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EMB_FUNC_STATUS _MAINPAGE	r	35h	IS_FSM_LC	0	-	-	-	0	0	0
FSM_STATUS_A _MAINPAGE	r	36h	IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1
FSM_STATUS_B _MAINPAGE	r	37h	IS_FSM16	IS_FSM15	IS_FSM14	IS_FSM13	IS_FSM12	IS_FSM11	IS_FSM10	IS_FSM9

3.1.1.1 *EMB_FUNC_STATUS_MAINPAGE (35h)*

The EMB_FUNC_STATUS_MAINPAGE (35h) register contains interrupt status information about the long counter.

Table 2. EMB_FUNC_STATUS_MAINPAGE (35h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM_LC	0	-	-	-	0	0	0

The IS_FSM_LC bit is automatically set to ‘1’ when the current long counter value, available in the embedded functions FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers, is equal to the long counter timeout value configured in the FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_ TIMEOUT_H (7Bh) registers.

3.1.1.2 *FSM_STATUS_A_MAINPAGE (36h)*

The FSM_STATUS_A_MAINPAGE (36h) register contains interrupt status information about programs 1-8.

Table 3. FSM_STATUS_A_MAINPAGE (36h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1

The IS_FSMx bit is set to ‘1’ when the OUTC / CONT / CONTREL command is performed in FSM program$_X$. Refer to the dedicated chapter / paragraph for additional details about these commands.

3.1.1.3 *FSM_STATUS_B_MAINPAGE (37h)*

The FSM_STATUS_B_MAINPAGE (37h) register contains interrupt status information about programs 9-16.

Table 4. FSM_STATUS_B_MAINPAGE (37h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM16	IS_FSM15	IS_FSM14	IS_FSM13	IS_FSM12	IS_FSM11	IS_FSM10	IS_FSM9

The IS_FSMx bit is set to ‘1’ when the OUTC / CONT / CONTREL command is performed in FSM program$_X$. Refer to the dedicated chapter / paragraph for additional details about these commands.

3.1.2 FSM embedded function registers

Table 5. Embedded function registers

Register name	Type	Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EMB_FUNC_EN_B	r/w	05h	0(1)	0(1)	0(1)	-	-	0(1)	0(1)	FSM_EN
EMB_FUNC_INT1	r/w	0Ah	INT1_FSM _LC(2)	0(1)	-	-	-	0(1)	0(1)	0(1)
FSM_INT1_A	r/w	0Bh	INT1_FSM8(2)	INT1_FSM7(2)	INT1_FSM6(2)	INT1_FSM5(2)	INT1_FSM4(2)	INT1_FSM3(2)	INT1_FSM2(2)	INT1_FSM1(2)
FSM_INT1_B	r/w	0Ch	INT1_FSM16(2)	INT1_FSM15(2)	INT1_FSM14(2)	INT1_FSM13(2)	INT1_FSM12(2)	INT1_FSM11(2)	INT1_FSM10(2)	INT1_FSM9(2)
EMB_FUNC_INT2	r/w	0Eh	INT2_FSM_LC(3)	0(1)	-	-	-	0(1)	0(1)	0(1)
FSM_INT2_A	r/w	0Fh	INT2_FSM8(3)	INT2_FSM7(3)	INT2_FSM6(3)	INT2_FSM5(3)	INT2_FSM4(3)	INT2_FSM3(3)	INT2_FSM2(3)	INT2_FSM1(3)
FSM_INT2_B	r/w	10h	INT2_FSM16(3)	INT2_FSM15(3)	INT2_FSM14(3)	INT2_FSM13(3)	INT2_FSM12(3)	INT2_FSM11(3)	INT2_FSM10(3)	INT2_FSM9(3)
EMB_FUNC_STATUS	r	12h	IS_FSM_LC	0	-	-	-	0	0	0
FSM_STATUS_A	r	13h	IS_FSM_8	IS_FSM_7	IS_FSM_6	IS_FSM_5	IS_FSM_4	IS_FSM_3	IS_FSM_2	IS_FSM_1
FSM_STATUS_B	r	14h	IS_FSM_16	IS_FSM_15	IS_FSM_14	IS_FSM_13	IS_FSM_12	IS_FSM_11	IS_FSM_10	IS_FSM_9
PAGE_RW	r/w	17h	EMB_FUNC_LIR	-	-	0	0	0	0	0
FSM_ENABLE_A	r/w	46h	FSM8_EN	FSM7_EN	FSM6_EN	FSM5_EN	FSM4_EN	FSM3_EN	FSM2_EN	FSM1_EN
FSM_ENABLE_B	r/w	47h	FSM16_EN	FSM15_EN	FSM14_EN	FSM13_EN	FSM12_EN	FSM11_EN	FSM10_EN	FSM9_EN
FSM_LONG_COUNTER_L	r	48h	FSM_LC7	FSM_LC6	FSM_LC5	FSM_LC4	FSM_LC3	FSM_LC2	FSM_LC1	FSM_LC0
FSM_LONG_COUNTER_H	r	49h	FSM_LC15	FSM_LC14	FSM_LC13	FSM_LC12	FSM_LC11	FSM_LC10	FSM_LC9	FSM_LC8
FSM_LONG_COUNTER_CLEAR	r/w	4Ah	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	FSM_LC _CLEARED(4)	FSM_LC _CLEAR
FSM_OUTS1	r	4Ch	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS2	r	4Dh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS3	r	4Eh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS4	r	4Fh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS5	r	50h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS6	r	51h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS7	r	52h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS8	r	53h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS9	r	54h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS10	r	55h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS11	r	56h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS12	r	57h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS13	r	58h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS14	r	59h	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS15	r	5Ah	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
FSM_OUTS16	r	5Bh	P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0
EMB_FUNC_ODR_CFG_B	r/w	5Fh	0(1)	1(5)	0(1)	FSM_ODR1	FSM_ODR0	0(1)	1(5)	1(5)
FSM_INIT	r/w	67h	0(1)	0(1)	0(1)	0(1)	-	0(1)	0(1)	FSM_INIT

1. *This bit must be set to ‘0’ for the correct operation of the device.*
2. *This bit is effective if INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to ‘1’.*
3. *This bit is effective if INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to '1'.*
4. *Read-only bit.*
5. *This bit must be set to '1' for the correct operation of the device.*

3.1.2.1 *EMB_FUNC_EN_B (05h)*

The EMB_FUNC_EN_B (05h) register is used to enable the FSM embedded functionality.

Table 6. EMB_FUNC_EN_B (05h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	-	-	0	0	FSM_EN

The FSM_EN bit is used to enable the FSM. When this bit is set to ‘1’, all enabled FSM programs start the execution.

3.1.2.2 *EMB_FUNC_INT1 (0Ah)*

The EMB_FUNC_INT1 (0Ah) register is used to route the FSM long counter interrupt on the INT1 pin: set the INT1_FSM_LC bit to ‘1’ in order to enable routing.

Table 7. EMB_FUNC_INT1 (0Ah) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT1_ FSM_LC	0	-	-	-	0	0	0

The INT1_FSM_LC bit is effective if the INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to ‘1’.

3.1.2.3 *FSM_INT1_A (0Bh)*

The FSM_INT1_A (0Bh) register is used for routing the FSM program 1-8 interrupts on the INT1 pin.

Table 8. FSM_INT1_A (0Bh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT1_FSM8	INT1_FSM7	INT1_FSM6	INT1_FSM5	INT1_FSM4	INT1_FSM3	INT1_FSM2	INT1_FSM1

These bits are effective if the INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to ‘1’.

Each bit on this register enables a signal to be carried on INT1. The pin’s output will supply the OR combination of the selected signals.

3.1.2.4 *FSM_INT1_B (0Ch)*

The FSM_INT1_B (0Ch) register is used for routing the FSM program 9-16 interrupts on the INT1 pin.

Table 9. FSM_INT1_B (0Ch) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT1_FSM16	INT1_FSM15	INT1_FSM14	INT1_FSM13	INT1_FSM12	INT1_FSM11	INT1_FSM10	INT1_FSM9

These bits are effective if the INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to ‘1’.

Each bit on this register enables a signal to be carried on INT1. The pin’s output will supply the OR combination of the selected signals.

3.1.2.5 *EMB_FUNC_INT2 (0Eh)*

The EMB_FUNC_INT2 (0Eh) register is used for routing the FSM long counter interrupt on the INT2 pin: set the INT2_FSM_LC bit to ‘1’ in order to enable routing.

Table 10. EMB_FUNC_INT2 (0Eh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT2_ FSM_LC	0	-	-	-	0	0	0

These bits are effective if the INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to ‘1’.

3.1.2.6 *FSM_INT2_A (0Fh)*

The FSM_INT2_A (0Fh) register is used for routing the FSM program 1-8 interrupts on the INT2 pin.

Table 11. FSM_INT2_A (0Fh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT2_FSM8	INT2_FSM7	INT2_FSM6	INT2_FSM5	INT2_FSM4	INT2_FSM3	INT2_FSM2	INT2_FSM1

These bits are effective if the INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to ‘1’.

Each bit on this register enables a signal to be carried on INT2. The pin’s output will supply the OR combination of the selected signals.

3.1.2.7 *FSM_INT2_B (10h)*

The FSM_INT2_B (10h) register is used for routing the FSM program 9-16 interrupts on the INT2 pin.

Table 12. FSM_INT2_B (10h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT2_FSM16	INT2_FSM15	INT2_FSM14	INT2_FSM13	INT2_FSM12	INT2_FSM11	INT2_FSM10	INT2_FSM9

These bits are effective if the INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to ‘1’.

Each bit on this register enables a signal to be carried on INT2. The pin’s output will supply the OR combination of the selected signals.

3.1.2.8 *EMB_FUNC_STATUS (12h)*

The EMB_FUNC_STATUS (12h) register contains interrupt status information about the long counter.

Table 13. EMB_FUNC_STATUS (12h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM_LC	0	-	-	-	0	0	0

The IS_FSM_LC bit is automatically set to ‘1’ when the current long counter value, available in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers, is equal to the long counter timeout value configured in the FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_ TIMEOUT_H (7Bh) registers.

3.1.2.9 *FSM_STATUS_A (13h)*

The FSM_STATUS_A (13h) register contains interrupt status information about programs 1-8.

Table 14. FSM_STATUS_A (13h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM8	IS_FSM7	IS_FSM6	IS_FSM5	IS_FSM4	IS_FSM3	IS_FSM2	IS_FSM1

The IS_FSMx bit is set to ‘1’ when the OUTC / CONT / CONTREL command is performed in FSM program$_X$. Refer to the dedicated chapter / paragraph for additional details about these commands.

3.1.2.10 *FSM_STATUS_B (14h)*

The FSM_STATUS_B (14h) register contains interrupt status information about programs 9-16.

Table 15. FSM_STATUS_B (14h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IS_FSM16	IS_FSM15	IS_FSM14	IS_FSM13	IS_FSM12	IS_FSM11	IS_FSM10	IS_FSM9

The IS_FSMx bit is set to ‘1’ when the OUTC / CONT / CONTREL command is performed in FSM program$_X$. Refer to the dedicated chapter / paragraph for additional details about these commands.

3.1.2.11 *PAGE_RW (17h)*

The PAGE_RW (17h) register is used to change the FSM interrupt from pulsed (default) to latched.

Table 16. PAGE_RW (17h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EMB_ FUNC_LIR	-	-	0	0	0	0	0

3.1.2.12 *FSM_ENABLE_A (46h)*

The FSM_ENABLE_A (46h) register is used for enabling programs 1-8 of the FSM.

Table 17. FSM_ENABLE_A (46h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM8_EN	FSM7_EN	FSM6_EN	FSM5_EN	FSM4_EN	FSM3_EN	FSM2_EN	FSM1_EN

3.1.2.13 *FSM_ENABLE_B (47h)*

The FSM_ENABLE_B (47h) register is used for enabling programs 9-16 of the FSM.

Table 18. FSM_ENABLE_B (47h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM16_EN	FSM15_EN	FSM14_EN	FSM13_EN	FSM12_EN	FSM11_EN	FSM10_EN	FSM9_EN

3.1.2.14 *FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h)*

The FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers are used to read / write the long counter value. Refer to Section 3.1 Configuration block for information about how to access these registers.

Table 19. FSM_LONG_COUNTER_L (48h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC7	FSM_LC6	FSM_LC5	FSM_LC4	FSM_LC3	FSM_LC2	FSM_LC1	FSM_LC0

Table 20. FSM_LONG_COUNTER_H (49h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC15	FSM_LC14	FSM_LC13	FSM_LC12	FSM_LC11	FSM_LC10	FSM_LC9	FSM_LC8

3.1.2.15 *FSM_LONG_COUNTER_CLEAR (4Ah)*

The FSM_LONG_COUNTER_CLEAR (4Ah) register is used to reset the FSM long counter value.

Table 21. FSM_LONG_COUNTER_CLEAR (4Ah) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0	0	0	FSM_LC_ CLEARED(1)	FSM_LC_ CLEAR

1. *Read-only bit.*

Set the FSM_LC_CLEAR bit to ‘1’ to reset the value of the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers the next time an INCR command is performed. When the long counter reset is done, the FSM_LC_CLEARED bit is automatically set to ‘1’. Refer to Section 5.1 Long Counter.

3.1.2.16 *FSM_OUTS[1:16] (4Ch - 5Bh)*

FSM[1:16] output register.

Table 22. FSM_OUTS[1:16] (4Ch - 5Bh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_X	N_X	P_Y	N_Y	P_Z	N_Z	0	0

These are read-only registers, one for each state machine, that contain the current active temporary mask value updated when the OUTC / CONT / CONTREL command is performed.

3.1.2.17 *EMB_FUNC_ODR_CFG_B (5Fh)*

The EMB_FUNC_ODR_CFG_B (5Fh) register is used to configure the ODR of the FSM (FSM_ODR[1:0] bits).

Table 23. EMB_FUNC_ODR_CFG_B (5Fh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	1	0	FSM_ODR1	FSM_ODR0	0	1	1

All the programs are executed at this configured rate. See Section 6.5 Decimator in Section 6 Variable Data section for information about how to run programs at different data rates.

Possible ODR configurations are listed in the following table.

Table 24. FSM output data rate

FSM_ODR[1:0]	ODR [Hz]
00	12.5
01	26
10	52
11	104

Note: The FSM ODR is internally limited to the accelerometer data rate. It is recommended to set the accelerometer data rate higher or equal to the configured FSM ODR.

3.1.2.18 *FSM_INIT (67h)*

The FSM_INIT (67h) register is used to reset the FSM programs to their default configuration.

Table 25. FSM_INIT (67h) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0	-	0	0	FSM_INIT

The FSM_INIT bit is used to trigger a new “Start Routine” request. When this bit is set to ‘1’, the device executes the start routine, described in Section 9 Start routine. When the start routine is completed, the FSM_INIT bit is automatically set to ‘0’.

In addition, this bit automatically goes to ‘1’ when the FSM_EN bit of EMB_FUNC_EN_B (05h) register is set to ‘0’ (and is reset to ‘0’ when the start routine is completed).

3.1.3 FSM embedded advanced features registers

The following table provides a list of the registers for the embedded advanced features pages 0 and 1 related to the FSM. These registers are accessible by configuring PAGE_SEL[3:0] bits in *PAGE_SEL (02h)*.

Table 26. FSM embedded advanced features registers

Register name	Page	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC_TIMEOUT_L	1	7Ah	FSM_LC_TIMEOUT 7	FSM_LC_TIMEOUT 6	FSM_LC_TIMEOUT 5	FSM_LC_TIMEOUT 4	FSM_LC_TIMEOUT 3	FSM_LC_TIMEOUT 2	FSM_LC_TIMEOUT 1	FSM_LC_TIMEOUT 0
FSM_LC_TIMEOUT_H	1	7Bh	FSM_LC_TIMEOUT 15	FSM_LC_TIMEOUT 14	FSM_LC_TIMEOUT 13	FSM_LC_TIMEOUT 12	FSM_LC_TIMEOUT 11	FSM_LC_TIMEOUT 10	FSM_LC_TIMEOUT 9	FSM_LC_TIMEOUT 8
FSM_PROGRAMS	1	7Ch	FSM_N_PROG7	FSM_N_PROG6	FSM_N_PROG5	FSM_N_PROG4	FSM_N_PROG3	FSM_N_PROG2	FSM_N_PROG1	FSM_N_PROG0
FSM_START_ADD_L	1	7Eh	FSM_START7	FSM_START6	FSM_START5	FSM_START4	FSM_START3	FSM_START2	FSM_START1	FSM_START0
FSM_START_ADD_H	1	7Fh	FSM_START15	FSM_START14	FSM_START13	FSM_START12	FSM_START11	FSM_START10	FSM_START9	FSM_START8

3.1.3.1 *FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh)*

The FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh) registers are used to set the long counter timeout register value.

Table 27. FSM_LC_TIMEOUT_L (7Ah) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC_ TIMEOUT7	FSM_LC_ TIMEOUT6	FSM_LC_ TIMEOUT5	FSM_LC_ TIMEOUT4	FSM_LC_ TIMEOUT3	FSM_LC_ TIMEOUT2	FSM_LC_ TIMEOUT1	FSM_LC_ TIMEOUT0

Table 28. FSM_LC_TIMEOUT_H (7Bh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_LC_ TIMEOUT16	FSM_LC_ TIMEOUT15	FSM_LC_ TIMEOUT14	FSM_LC_ TIMEOUT13	FSM_LC_ TIMEOUT12	FSM_LC_ TIMEOUT11	FSM_LC_ TIMEOUT10	FSM_LC_ TIMEOUT9

3.1.3.2 *FSM_PROGRAMS (7Ch)*

The FSM_PROGRAMS (7Ch) register is used to set the number of configured state machines.

Table 29. FSM_N_PROG (7Ch) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_N_ PROG7	FSM_N_ PROG6	FSM_N_ PROG5	FSM_N_ PROG4	FSM_N_ PROG3	FSM_N_ PROG2	FSM_N_ PROG1	FSM_N_ PROG0

This register must be configured coherently with configured state machines for the correct operation of the device. The maximum allowed value is 16 (0x10).

3.1.3.3 *FSM_START_ADD_L (7Eh) and FSM_START_ADD_H (7Fh)*

The FSM_START_ADD_L (7Eh) and FSM_START_ADD_H (7Fh) registers are used to set the FSM program start address.

Table 30. FSM_START_ADD_L (7Eh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_START7	FSM_START6	FSM_START5	FSM_START4	FSM_START3	FSM_START2	FSM_START1	FSM_START0

For the correct operation of the device, the value of this register must be set equal to '00h' if not configured differently by the Unico tool.

Table 31. FSM_START_ADD_H (7Fh) register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FSM_START16	FSM_START15	FSM_START14	FSM_START13	FSM_START12	FSM_START11	FSM_START10	FSM_START9

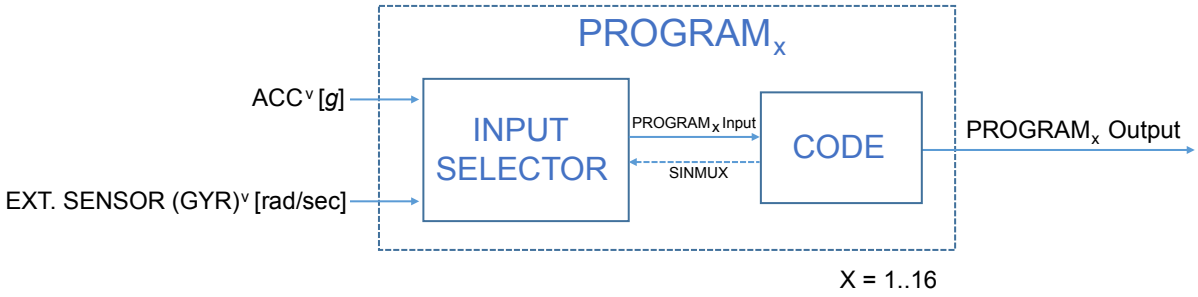
For the correct operation of the device, the value of this register must be set equal to '04h' if not configured differently by the Unico tool.

3.2 Program block

Output data coming from the Signal Conditioning block are sent to the FSM block, composed of 16 Program blocks. Each Program block, as shown in the following figure, consists of:

- an Input Selector block, that selects the desired input data signal that will be processed by the program;
- a Code block, composed of the data and the instructions that will be executed.

Figure 5. Program block



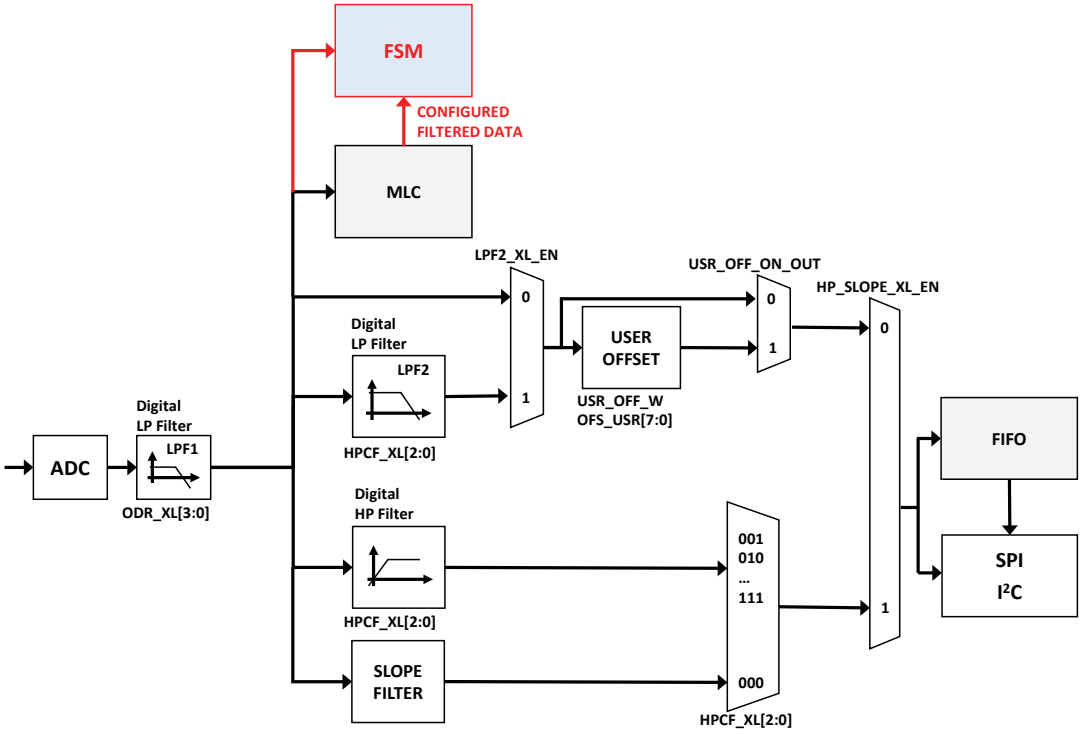
3.2.1 Input Selector block

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

- IIS2ICLX accelerometer data;
- external sensor (e.g. gyroscope) data;
- internally filtered data, by properly configuring the Machine Learning Core;

The Machine Learning Core allows configuring high-pass, band-pass, IIR1 and IIR2 filters applied to sensor data.
The following figure shows the inputs of the Finite State Machine block in the accelerometer digital chain.

Figure 6. FSM inputs



The signal bandwidth of the accelerometer depends on the device configuration. For additional information, please refer to AN5509 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.

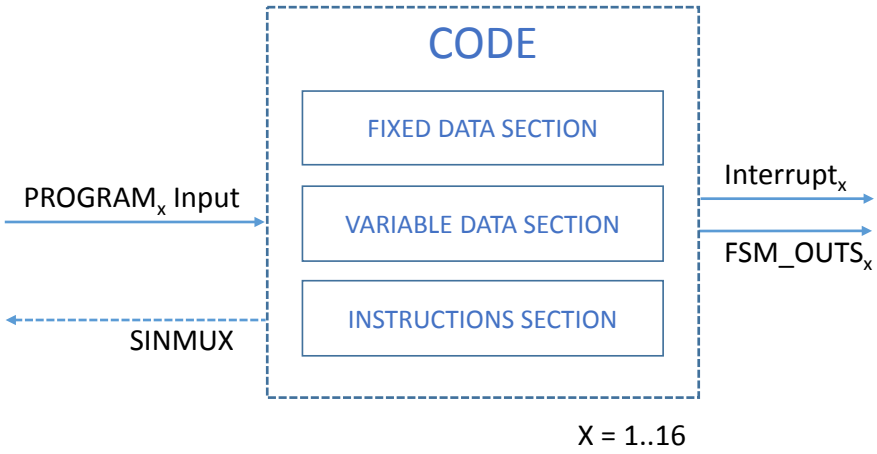
3.2.2 Code block

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 Instruction 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.

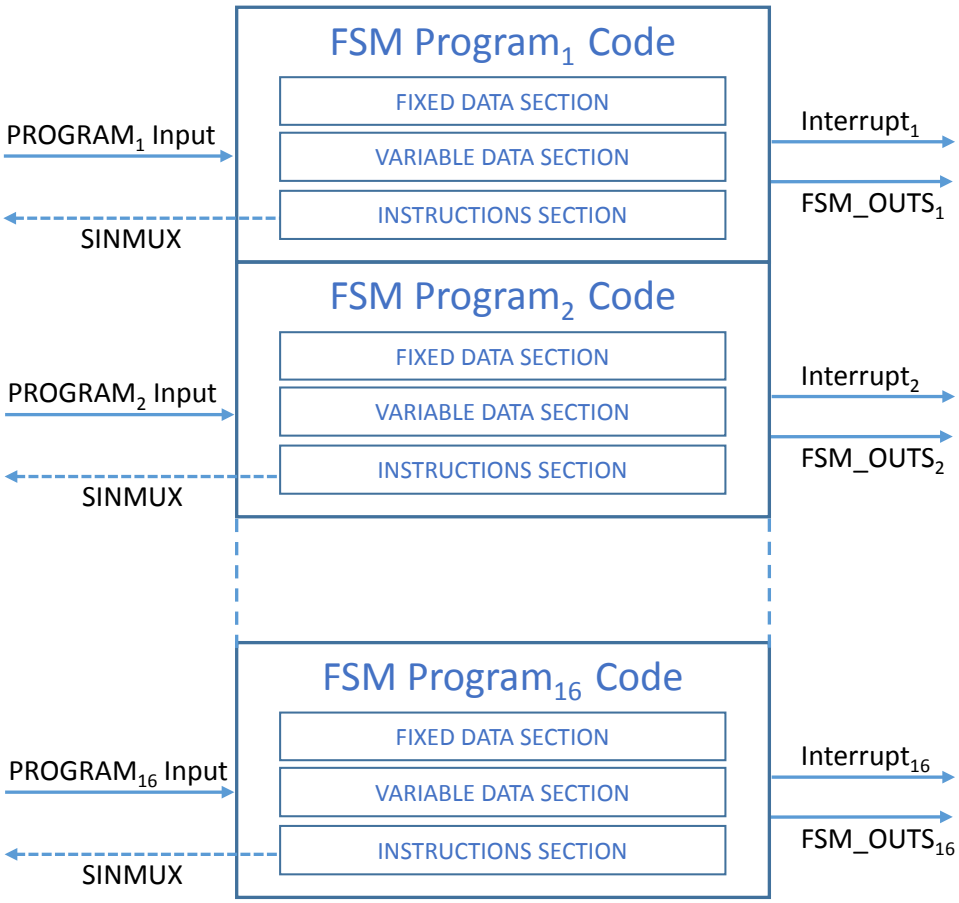
Figure 7. FSM Program$_x$ Code structure



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 have to be reconfigured each time the device is powered on.

Figure 8. **FSM Program$_x$ memory area**



4 FSM Interrupt

The FSM interrupt signal is generated when the end state is reached or when some specific command is performed (OUTC / CONT / CONTREL commands). When an 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/H embedded function register, reaches the configured long counter timeout value in FSM_LC_TIMEOUT_L/H embedded advanced features register (page 1).

The FSM interrupt and the FSM long counter interrupt signals can be checked by reading the dedicated register:

- EMB_FUNC_STATUS_MAINPAGE (35h) register or EMB_FUNC_STATUS (12h) embedded function register for the long counter interrupt status;
- FSM_STATUS_A_MAINPAGE (36h) or FSM_STATUS_B_MAINPAGE (37h) registers or FSM_STATUS_A (13h) or FSM_STATUS_B (14h) embedded function registers for the FSM interrupt status.

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

- INT1_FSM_LC/INT2_FSM_LC bit of the EMB_FUNC_INT1/EMB_FUNC_INT2 embedded function register to 1;
- INT1_FSM[1:16]/INT2_FSM[1:16] bit of the FSM_INT1_A/FSM_INT1_B/FSM_INT2_A/FSM_INT2_B embedded function register to 1;

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

The behavior of the interrupt signal is pulsed by default. The duration of the pulse is equal to 1/ODR_XL.

Note: Minimum pulse duration is 1/104 Hz (~9.6 msec).

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 is reset by reading:

- EMB_FUNC_STATUS_MAINPAGE (35h) register or EMB_FUNC_STATUS (12h) embedded function register for the long counter interrupt status;
- FSM_STATUS_A_MAINPAGE (36h) or FSM_STATUS_B_MAINPAGE (37h) registers or FSM_STATUS_A (13h) or FSM_STATUS_B (14h) embedded function registers for the FSM interrupt status.

5 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	HYST	-	PAS	DECTREE	STOPDONE	LC	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 3 thresholds (NR_THRESH bits);
  - up to 3 masks (NR_MASK bits);
  - up to 2 long (16 bits) timers (NR_LTIMER bits);
  - up to 2 short (8 bits) timers (NR_TIMER bits).
- With CONFIG_B it is possible to declare:
  - a decimation factor for incoming ODR (DES bit);
  - a hysteresis value (HYST bit);
  - usage of previous axis signs, that have to be computed and stored (PAS bit);
  - usage of a decision tree interface (DECTREE bit);
  - usage of the long counter, common to all state machines (LC 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, etc…).
- 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 9 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.

5.1 Long Counter

The long counter is a temporary counter resource available to the user; it's possible to increment its value, stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers, by using the INCR command.

This resource is common to all programs and does not need additional allocated resources in the Variable Data section. In order to use the long counter resource, the LC bit of CONFIG_B byte must be set to '1'.

When the long counter value (FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers) is equal to the configured long counter timeout value (FSM_LC_TIMEOUT_L (7Ah) and FSM_LC_TIMEOUT_H (7Bh) registers), the IS_FSM_LC status bit of the EMB_FUNC_STATUS (12h) register is set to '1'.

It is possible to route this signal to the:

- INT1 pin if both the:
  - INT1_FSM_LC bit of the EMB_FUNC_INT1 (0Ah) register is set to 1;
  - INT1_EMB_FUNC bit of the MD1_CFG (5Eh) is set to '1'.
- INT2 pin if both the:
  - INT2_FSM_LC bit of the EMB_FUNC_INT2 (0Eh) register is set to 1;
  - INT2_EMB_FUNC bit of the MD2_CFG (5Fh) is set to '1'.

The IS_FSM_LC bit is cleared by reading the EMB_FUNC_STATUS_MAINPAGE (35h) register or the EMB_FUNC_STATUS (12h) embedded functions register. However, if the FSM_LONG_COUNTER value is not reset, an interrupt will be generated each time an INCR command is issued.

In order to properly reset the FSM long counter, the user has to set the FSM_LC_CLEAR bit of the FSM_LONG_COUNTER_CLEAR (4Ah) register to '1'. In this case, the next time an INCR command is performed, the reset procedure starts. When this reset procedure is completed, the FSM_LC_CLEARED bit of the FSM_LONG_COUNTER_CLEAR (4Ah) register is automatically set to '1'. Finally, the FSM_LC_CLEAR bit of the FSM_LONG_COUNTER_CLEAR (4Ah) register bit has to be manually reset to '0'.

6 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 7	THRESH1	THRESH1(15:0)							
8 9	THRESH2	THRESH2(15:0)							
10 11	THRESH3	THRESH3(15:0)							
12 13	HYST	HYSTERESIS(15:0)							
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 21	TC	TC(15:0) or TC(7:0)							
22 23	TIMER1	TIMER1(15:0)							
24 25	TIMER2	TIMER2(15:0)							
26	TIMER3	TIMER3(7:0)							
27	TIMER4	TIMER4(7:0)							
28	DEST	DEST(7:0)							
29	DESC	DESC(7:0)							
30	PAS	SCTC	-	MSKIT	MSKITEQ	SIGN_X	SIGN_Y	SIGN_Z	-
31	DECTREE	-	DTSEL(2:0)			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), 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).

6.1 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 IIS2ICLX accelerometer signal is selected, the unit of the threshold is [*g*]. In this case, the threshold value to be written in the device threshold registers has to be multiplied by 4 before its float32 to float16 conversion: for example, if a threshold of 1[*g*] is intended to be used, the value to be reported in the threshold registers is 4400h, which corresponds to 4;
- if the external gyroscope sensor signal is selected, the unit of the threshold is [rad/sec];
- if the MLC filter signal is selected, the unit of the threshold is the same as that of the filtered signal (it can be [*g*] for the accelerometer or [rad/sec] for an external gyroscope).

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, 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.

6.2 Hysteresis

The hysteresis resource affects the current threshold value when a threshold comparison is performed. If the hysteresis resource is used, the hysteresis value is automatically:

- added to the threshold used in all “GREATER THAN” conditions (GNTH1, GNTH2, GLTH1 and GRTH1);
- subtracted from the threshold used in all “LESS THAN” conditions (LNTH1, LNTH2, LLTH1 and LRTH1).

Examples:

- if “GNTH1” condition is performed, the threshold used is: THRESH1 + Hysteresis;
- if “LNTH2” condition is performed, the threshold used is: THRESH2 – Hysteresis.

By setting the HYST bit of CONFIG_B byte to ‘1’, the HYSTERESIS[15:0] resource can be properly configured in the Variable Data section.

Involved commands:

- N/A.

Involved conditions:

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

Note: Hysteresis does not affect zero-crossing conditions.

6.3 Masks / temporary masks

Mask resources are used to enable or disable mask action on the input data (X, Y, Z) 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. 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	0	0

Note: The IIS2ICLX works as a 2-axis linear accelerometer sensor which provides data on the X and Y axes. When the IIS2ICLX accelerometer signal is selected, the +Z and -Z bits must be set to '0'.

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), that will be 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 = 90h (10010000b) → +X and -Y are enabled;
- Current input accelerometer sample = [0.72 -0.45 0 0].

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

It is possible to reset the temporary mask value to the mask value in 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;
- SWAPMSK;
- SISW.

Involved conditions:

- GNTH1 / GNTH2 / GLTH1 / GRTH1;

- LNTH1 / LNTH2 / LLTH1 / LRTH1;
- PZC / NZC.

6.4 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[1:0] bits of the EMB_FUNC_ODR_CFG_B (5Fh) register, including the decimation factor if used. Long timer resources are called TI1 and TI2, while short timer resources are called TI3 and TI4. An additional internal Timer Counter (TC) is used as temporary counter to check if a timer has elapsed. The TC value can be preloaded with two different modalities, selectable by using the SCTC0 / SCTC1 commands:

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

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

Example:

- Timer TI3 is set equal to 10. 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;
- 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;
- 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’, TC resource is not allocated;
- if NR_LTIMER[1:0] = ‘00’ and NR_TIMER[1:0] ≠ ‘00’, TC resource occupies one byte;
- if NR_LTIMER[1:0] ≠ ‘00’ and NR_TIMER[1:0] = ‘00’, TC resource occupies two bytes;
- if NR_LTIMER[1:0] ≠ ‘00’ and NR_TIMER[1:0] ≠ ‘00’, TC resource occupies two bytes;

Involved commands:

- STIMER3 / STIMER4;
- SCTC0 / SCTC1.

Involved conditions:

- TI1 / TI2 / TI3 / TI4.

6.5 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[1:0] bits of the EMB_FUNC_ODR_CFG_B (5Fh) register, according to following formula:

PROGRAM_ODR = FSM_ODR / DEST

At startup:

DESC = DEST (initial decimation value)

when 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.

Commands involved:

- N/A.

Conditions involved:

- N/A.

Note: The minimum meaningful value for DEST is ‘2’.

6.6 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.

6.7 Decision Tree interface

The Decision Tree interface resource is accessible by using the CHKDT condition, that can be used to check the result of one of the eight decision trees available inside the Machine Learning Core algorithms. This can be very useful when a machine learning logic is expected to be combined with an FSM program.

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(2:0) bits, from 0 to 7) and the corresponding expected value (DTRES(3:0) bits, from 0 to 15).

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.

Involved commands:

- N/A

Involved conditions:

- CHKDT

Note: It is not possible to perform the following conditions:

- PZC | CHKDT opcode (0xDF) is equal to SMB opcode;

- CHKDT | NZC opcode (0xFE) is equal to SMC opcode;

- NZC | CHKDT opcode (0xEF) is equal to MSKITEQ opcode;

- CHKDT | GNTH1 opcode (0xF5) is equal to MSKIT opcode.

7 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.

7.1 Reset/Next conditions

RESET/NEXT conditions are used to reset or continue the program flow. RESET/NEXT conditions are executed in one single state when a new sample set is ready.

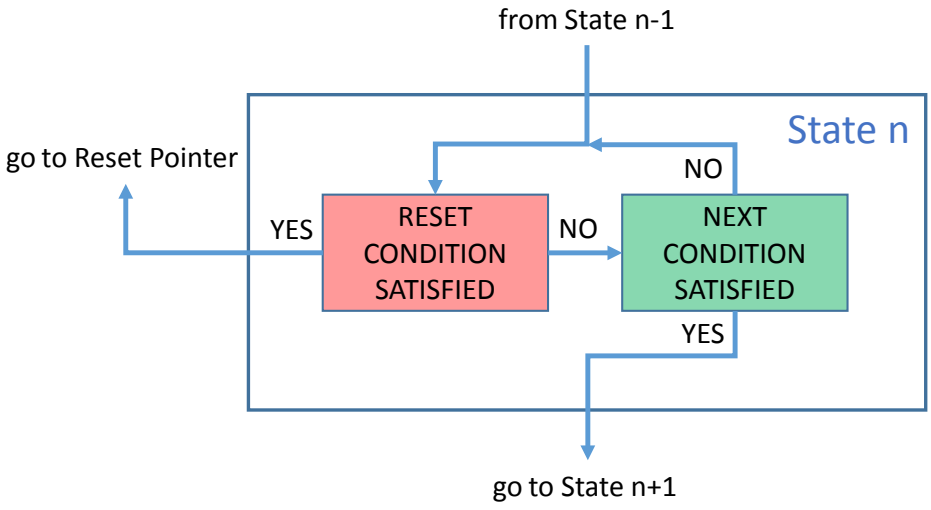
The RESET condition is defined in the opcode MSB part while the NEXT condition is defined in the opcode LSB part. As shown in the following figure, the RESET condition is always performed before the NEXT condition, that is evaluated only when the RESET condition is not satisfied.

When both conditions (NEXT and RESET) are not satisfied, the state machine waits for a new sample set (X, Y, Z) and starts the evaluation again in the same state.

A transition to the reset pointer occurs whenever the “RESET condition” is true (PP = RP).

A transition to the next step occurs whenever the “RESET condition” is false and “NEXT condition” is true and (PP = PP + 1).

Figure 11. Single state description



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 sixteen different conditions can be coded: the list of available conditions is shown in the following table. There are three 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 accelerometer X, Y, Z axis are higher (or lower) than a programmed threshold;
- zero-crossing detection: these conditions are true when an enabled input crosses the zero level.

Table 32. 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 for the NEXT and the RESET condition 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.

7.1.1 NOP (0h)

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

Actions:

- If NOP is in RESET condition: when a new sample set is ready, evaluates only the NEXT condition;
- If NOP is in NEXT condition: when a new sample set is ready, evaluates only the RESET condition.

7.1.2 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 sample set (X, Y, Z) occurs, then TC = TC – 1:
  - If TC > 0, continue comparisons in the current state (wait for new samples);
  - If TC = 0, the condition is valid:
    - If TI1 is in RESET position, PP = RP;
    - If TI1 is in NEXT position, PP = PP + 1.

7.1.3 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 sample set (X, Y, Z) occurs, then TC = TC – 1:
  - If TC > 0, continue comparisons in the current state (wait for new samples);
  - If TC = 0, the condition is valid:
    - If TI2 is in RESET position, PP = RP;
    - If TI2 is in NEXT position, PP = PP + 1.

7.1.4 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 sample set (X, Y, Z) occurs, then TC = TC – 1:
  - If TC > 0, continue comparisons in the current state (wait for new samples);
  - If TC = 0, the condition is valid:
    - If TI3 is in RESET position, PP = RP;
    - If TI3 is in NEXT position, PP = PP + 1.

7.1.5 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 sample set (X, Y, Z) occurs, then TC = TC – 1:
  - If TC > 0, continue comparisons in the current state (wait for new samples);
  - If TC = 0, the condition is valid:
    - If TI4 is in RESET position, PP = RP;
    - If TI4 is in NEXT position, PP = PP + 1.

7.1.6 GNTH1 (5h)

Description: GNTH1 condition is valid if any triggered axis of the data sample set (X, Y, Z) is greater than threshold 1 level. Threshold is: THRESH1 + HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in the variable data section is not '0'.

Action:

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

7.1.7 GNTH2 (6h)

Description: GNTH2 condition is valid if any triggered axis of the data sample set (X, Y, Z) is greater than threshold 2 level. Threshold is: THRESH2 + HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in the variable data section is not '0'.

Action:

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

7.1.8 LNTH1 (7h)

Description: LNTH1 condition is valid if any triggered axis of the data sample set (X, Y, Z) is lower than or equal to threshold 1 level. Threshold is: THRESH1 - HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.9 LNTH2 (8h)

Description: LNTH2 condition is valid if any triggered axis of the data sample set (X, Y, Z) is lower than or equal to threshold 2 level. Threshold is: THRESH2 - HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.10 GLTH1 (9h)

Description: GLTH1 condition is valid if all axes of the data sample set (X, Y, Z) are greater than threshold 1 level. Threshold is: THRESH1 + HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.11 LLTH1 (Ah)

Description: LLTH1 condition is valid if all axes of the data sample set (X, Y, Z) are less than or equal to threshold 1 level. Threshold is: THRESH1 - HYST.

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.12 GRTH1 (Bh)

Description: GRTH1 condition is valid if any triggered axis of the data sample set (X, Y, Z) is greater than threshold 1 level. Threshold is: – (THRESH1 + HYST).

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.13 LRTH1 (Ch)

Description: LRTH1 condition is valid if any triggered axis of the data sample set (X, Y, Z) is less than or equal to threshold 1 level. Threshold is: – (THRESH1 – HYST).

Note: The HYST value is involved in the threshold comparison only if the CONFIG_A(HYST) bit of the fixed data section is set to '1' and the HYST value in variable data section is not '0'.

Action:

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

7.1.14 PZC (Dh)

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

Action:

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

7.1.15 NZC (Eh)

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

Action:

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

7.1.16 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 6.7 Decision Tree interface.

Action:

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

7.2 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, that is immediately evaluated:

- if new line is a command, it is immediately executed again;
- if new line is a condition, it will be executed when the next sample is processed.

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

“AAh” (STHR1 command)

“CDh” (1st parameter)

“3Ch” (2nd parameter)

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

Table 33. 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
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
EEh	SISW	Swaps sign information to opposite in selected mask	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	Byte 1: input value for multiplexer

Opcode	Mnemonic	Description	Parameter
24h	STIMER3	Set new value to TIMER3 register	Byte 1: TI3 value
31h	STIMER4	Set new value to TIMER4 register	Byte 1: TI4 value
32h	SWAPMSK	Swap mask selection MASKA <=> MASKB; MASKC unaffected	None
34h	INCR	Increase long counter +1, check long counter timeout and clear	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	Don't 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

7.2.1 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, accordingly with use of MSKIT / MSKITEQ / UMSKIT commands);
- Stops itself by setting the CONFIG_B(STOPDONE) bit of the fixed data section to '1'. The user should disable and enable the corresponding state machine bit in the FSM_ENABLE_A (46h) or FSM_ENABLE_B (47h) register to restart the program. In this case, the Start Routine is performed. For additional information about the Start Routine refer to Section 9 Start routine.

7.2.2 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.

7.2.3 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, accordingly with use of MSKIT / MSKITEQ / UMSKIT commands);
- PP = RP.

7.2.4 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.

7.2.5 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.

7.2.6 SETP (55h)

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

Parameters: two bytes.

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

Actions:

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

7.2.7 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.

7.2.8 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.

7.2.9 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

7.2.10 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, accordingly with use of MSKIT / MSKITEQ / UMSKIT commands);
- PP = PP + 1.

7.2.11 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.

7.2.12 STHR2 (BBh)

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

7.2.13 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.

7.2.14 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.

7.2.15 SISW (EEh)

Description: SISW command swaps the temporary axis mask sign to the opposite sign.

Parameters: None.

Actions:

- Changes selected temporary mask axis sign to the opposite:
  - If sign(axis) is positive, new sign(axis) is negative;
  - If sign(axis) is negative, new sign(axis) is positive;
  - If axis information is zero, no changes.
- PP = PP + 1.

7.2.16 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.

7.2.17 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.

7.2.18 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.

7.2.19 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.

7.2.20 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.

7.2.21 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 SINMUX command can be also used to select the MLC filtered data; for this purpose, the first MLC filter has to be applied to the sensor axes.

Parameters: one byte.

- 1st parameter: value to select input source:

  0: accelerometer [a_x a_y 0 0];

  2: external gyroscope [g_x g_y g_z 0];

  3: filtered signal from Machine Learning Core(1) [F_x F_y F_z 0].

Actions:

- Selects input signal accordingly with set parameter. It configures the SETTINGS(IN_SEL[2:0]) bits of the fixed data section accordingly to the selected input source signal (it can be 000b, 010b or 011b);
- PP = PP + 2.

(1) Filter type could be HP / LP / IIR1 / IIR2 depending on the Machine Learning Core configuration.

Note: If the MLC is configured to filter the IIS2ICLX accelerometer signal, F_z is 0.

7.2.22 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.

7.2.23 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.

7.2.24 SWAPMSK (32h)

Description: SWAPMSK command is used to swap MASKA and MASKB selection. MASKC is not affected.

Parameters: None.

Actions:

- Swaps MASKA with MASKB;
- PP = PP + 1.

7.2.25 INCR (34h)

Description: INCR command is used to reset the long counter if the FSM_LC_CLEAR bit of the FSM_LONG_COUNTER_CLEAR (4Ah) register is set to '1', or to increase the long counter value by one. The long counter value is stored in the FSM_LONG_COUNTER_L (48h) and FSM_LONG_COUNTER_H (49h) registers.

Parameters: None.

Actions:

- Resets the long counter value if the FSM_LC_CLEAR bit of FSM_LONG_COUNTER_CLEAR (4Ah) register is set to '1', or increases the long counter value by one;
- PP = PP + 1.

7.2.26 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 CONFIG_A byte).

Actions:

- It sets to '1' the CONFIG_B(JMP) bit of the fixed data section. Evaluates the "NEXT1 | NEXT2" condition:
  - If "NEXT1" condition is true, PP = 2nd parameter address;
  - Else if "NEXT2" condition is true, PP = 3rd parameter address;
  - Else waits for a new sample set and evaluates again the "NEXT1 | NEXT2" condition.

7.2.27 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.

7.2.28 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.

7.2.29 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.

7.2.30 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.

7.2.31 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.

7.2.32 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.

7.2.33 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.

7.2.34 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.

8 FSM configuration example

This section contains an example that explains all write operations that have to be done in order to configure the IIS2ICLX FSM. A few steps have to be followed:

- configure the MLC to apply a high-pass filter on the IIS2ICLX accelerometer X and Y axes;
- configure the FSM registers inside the embedded function registers set;
- configure the FSM registers inside the embedded advanced features registers set;
- configure the IIS2ICLX accelerometer sensor.

In this example, two simple programs are configured:

- PROGRAM 1: tilt (around the x-axis) algorithm, routed on the INT1 pin;
- PROGRAM 2: wake-up algorithm, based on high-pass filtered data, routed on the INT2 pin.

Both algorithms are intended to use accelerometer data only at a sample rate of 26 Hz.

Refer to the figure below for details about the program data section and the instructions section.

Figure 12. FSM configuration example

	PAGE - ADDRESS	NAME	7	6	5	4	3	2	1	0
PROGRAM 1	4 - 00h	CONFIG A	01 (1 threshold)		01 (1 mask)		0		01 (1 short timer)	
	4 - 01h	CONFIG B	0	0	-	0	0	0	0	0
	4 - 02h	SIZE	10h (16 bytes)							
	4 - 03h	SETTINGS	0		0	0	0	0		
	4 - 04h	RESET POINTER	00h							
	4 - 05h	PROGRAM POINTER	00h							
	4 - 06h	THRESH1	BFAEh (-0.480)							
	4 - 07h									
	4 - 08h	MASKA	80h (+X)							
	4 - 09h	TMASKA	00h							
	4 - 0Ah	TC	00h							
	4 - 0Bh	TIMER3	10h (16 samples)							
	4 - 0Ch	GNTH1 | TI3	53h							
	4 - 0Dh	OUTC	99h							
	4 - 0Eh	GNTH1 | NOP	50h							
	4 - 0Fh	STOP	00h							
PROGRAM 2	4 - 10h	CONFIG A	01 (1 threshold)		01 (1 mask)		0		0	
	4 - 11h	CONFIG B	0	0	-	0	0	0	0	0
	4 - 12h	SIZE	10h (16 bytes)							
	4 - 13h	SETTINGS	0		0	0	0	0		
	4 - 14h	RESET POINTER	00h							
	4 - 15h	PROGRAM POINTER	00h							
	4 - 16h	THRESH1	3266h (0.05)							
	4 - 17h									
	4 - 18h	MASKA	A0h (+X and +Y)							
	4 - 19h	TMASKA	00h							
	4 - 1Ah	SINMUX	23h							
	4 - 1Bh	3	03h							
	4 – 1Ch	SSIGN0	12h							
	4 – 1Dh	SRP	33h							
	4 – 1Eh	NOP | GNTH1	05h							
	4 – 1Fh	CONTRL	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 10h // Set accelerometer sensor in power-down mode
2. Write 80h to register 01h // Enable access to embedded function registers

3. Write 01h to register 05h // EMB_FUNC_EN_B(FSM_EN) = '1'
4. Write 4Bh to register 5Fh // EMB_FUNC_ODR_CFG_B (FSM_ODR) = '01' (26Hz)
5. Write 03h to register 46h // FSM_ENABLE_A = '03h'
6. Write 00h to register 47h // FSM_ENABLE_B = '00h'
7. Write 01h to register 0Bh // FSM_INT1_A = '01h'
8. Write 00h to register 0Ch // FSM_INT1_B = '00h'
9. Write 02h to register 0Fh // FSM_INT2_A = '02h'
10. Write 00h to register 10h // FSM_INT2_B = '00h'
11. Write 40h to register 17h // PAGE_RW: enable write operation
12. Write 11h to register 02h // Enable access to embedded advanced features registers, PAGE_SEL = 1
13. Write 7Ah to register 08h // PAGE_ADDRESS = 7Ah
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 02h to register 09h // Dummy write in order to increment the write address
18. Write 00h to register 09h // Write 00h to register FSM_START_ADDRESS_L
19. Write 04h to register 09h // Write 04h to register FSM_START_ADDRESS_H
20. Write 41h to register 02h // PAGE_SEL = 4
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 00h to register 09h // RESET POINTER
27. Write 00h to register 09h // PROGRAM POINTER
28. Write AEh to register 09h // THRESH1 LSB
29. Write BFh 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 10h to register 09h // SIZE
41. Write 00h to register 09h // SETTINGS
42. Write 00h to register 09h // RESET POINTER
43. Write 00h to register 09h // PROGRAM POINTER
44. Write 66h to register 09h // THRESH1 LSB
45. Write 32h to register 09h // THRESH1 MSB

46. Write A0h to register 09h // MASKA
47. Write 00h to register 09h // TMASKA
48. Write 23h to register 09h // SINMUX
49. Write 03h to register 09h // Select MLC filter
50. Write 12h to register 09h // SSIGN0
51. Write 33h to register 09h // SRP
52. Write 05h to register 09h // NOP | GNTH1
53 Write 22h to register 09h // CONTREL
54. Write 01h to register 02h // Disable access to embedded advanced features registers, PAGE_SEL = 0
55. Write 00h to register 17h // PAGE_RW: disable write operation
56. Write 00h to register 01h // Disable access to embedded function registers
57. Write 02h to register 5Eh // MD1_CFG(INT1_EMB_FUNC) = '1'
58. Write 02h to register 5Fh // MD2_CFG(INT2_EMB_FUNC) = '1'
59. Write 2Ch to register 10h // CTRL1_XL = '2Ch' (26 Hz, ±2 *g*)

9 Start routine

When the FSM is enabled, a start routine is automatically executed. This routine performs the following tasks:

- the CONFIG_B(STOPDONE) and CONFIG_B(JMP) bits are reset;
- the PP and RP pointers are initialized to the first line of code;
- the SETTINGS field is initialized with 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 previous axis sign is declared, it is initialized to 0 (PAS = 0);
- if CONFIG_B(LC) is active, the long counter is reset.

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’).

10 Examples of state machine configurations

10.1 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 13. 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
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.

10.2 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 a motion intensity algorithm able to detect three intensity levels: stationary, low-intensity, high-intensity;
- An output value is associated to each recognized motion intensity level:
  - "Stationary" output is 0;
  - "Low-intensity" output is 1;
  - "High-intensity" output is 2;
- The window length for the features calculation is 2 seconds (52 samples having an ODR equal to 26 Hz).

The FSM is configured to generate an interrupt when the "High-intensity" level is detected.

Figure 14. Decision tree interface example

BYTE #	NAME	7	6	5	4	3	2	1	0
00h	CONFIG A	00 (0 threshold)		00 (0 mask)		00		00 (0 short timer)	
01h	CONFIG B	0	0	-	0	1	0	0	0
02h	SIZE	0Ah (10 bytes)							
03h	SETTINGS	00		0	0	0	00		
04h	RESET POINTER	00h							
05h	PROGRAM POINTER	00h							
06h	DECTREE	02h (selected decision tree number ‘0’, expected output is ‘2’)							
07h	NOP | CHKDT	0Fh							
08h	CONTREL	22h							
09h	STOP	00h							

Instructions section description

PP = 07h: 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 ‘2’ (i.e. "High-intensity"). If the detected motion intensity is "High-intensity", then the PP is increased (PP = PP + 1).

PP = 08h: CONTREL command is performed without needing a sample set: this generates an interrupt and resets the program (PP = RP = 07h).

11 Finite State Machine tool

The Finite State Machine programmability in the device is allowed through a dedicated tool, available as an extension of the Unico GUI.

11.1 Unico GUI

Unico 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-MKI109V3.

Unico GUI is available in three software packages for the three operating systems supported.

- Windows
  - STSW-MKI109W
- Linux
  - STSW-MKI109L
- Mac OS X
  - STSW-MKI109M

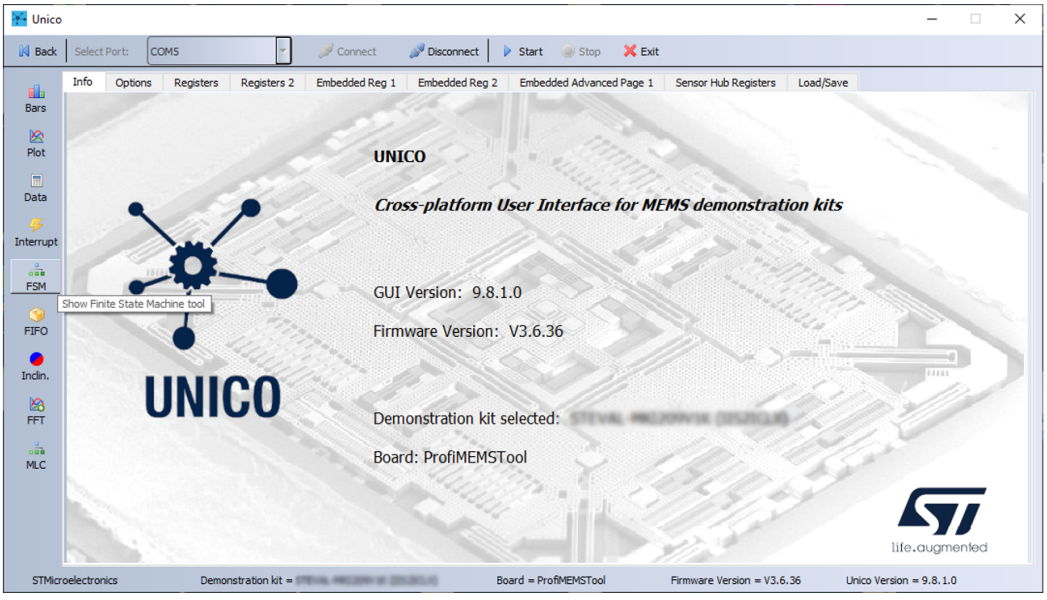
Unico GUI allows visualization of sensor outputs in both graphical and numerical format and allows the user to save or generally manage data coming from the device.

Unico 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 text file and load a configuration from an existing file. In this way, the sensor can be re-programmed in few seconds.

The Finite State Machine tool available in the Unico GUI 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.

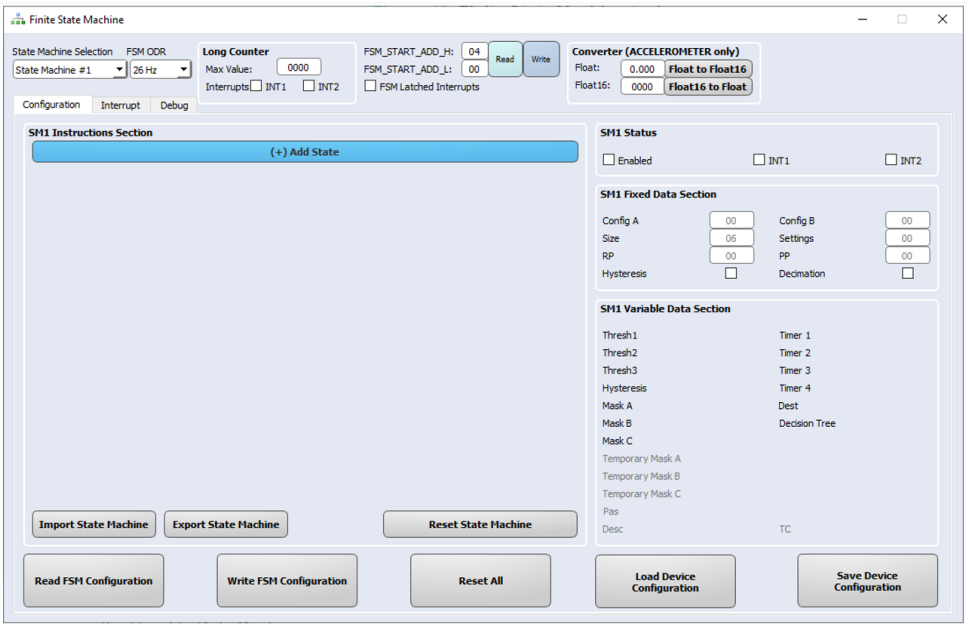
To execute the Finite State Machine tool, the user has to click on the dedicated “FSM” button that is available in the left side of the main UNICO GUI window as shown in the following figure.

Figure 15. Running the Finite State Machine tool



When loaded, the main Finite State Machine tool window is shown.

Figure 16. 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, the long counter parameters and the FSM latched interrupts. The FSM start address is automatically managed by the Unico tool and should not be changed by the user. Finally, a converter from float32 to float16 format and vice versa is available: the factor of '4' described in Section 6.1 Thresholds is automatically managed by the converter. The converter is used to generate the value to be set in the threshold resources in the Variable Data Section.

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

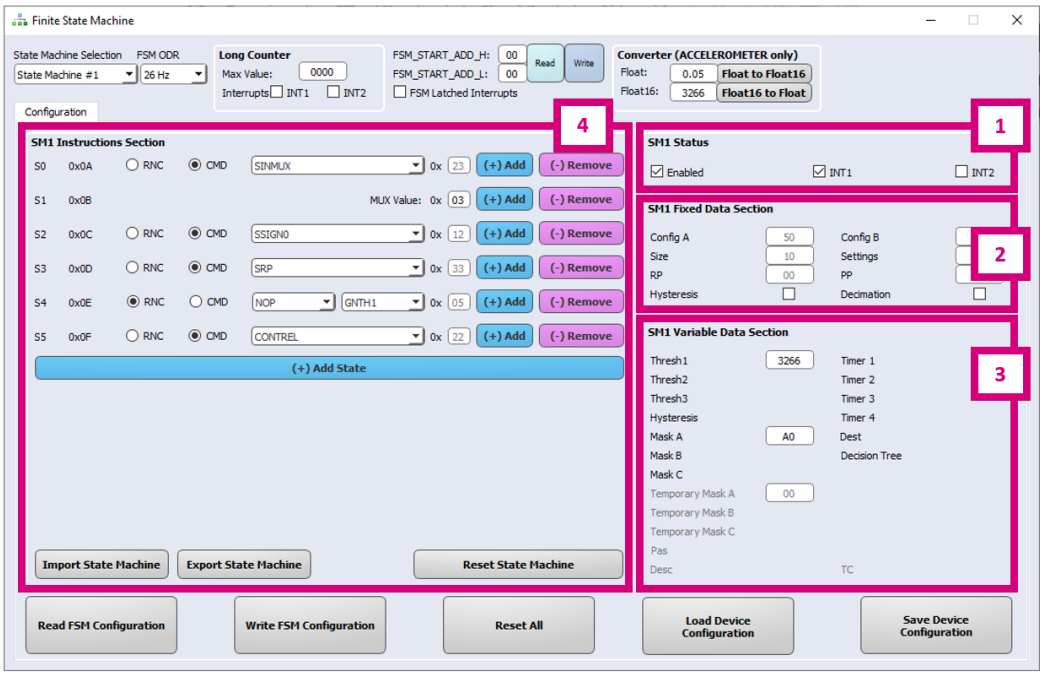
- Configuration tab (the one selected by default);
- Interrupt tab;
- Debug tab.

11.1.1 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, 4 group boxes are shown:

1. SM_x Status;
2. SM_x Fixed Data Section;
3. SM_x Variable Data Section;
4. SM_x Instructions Section.

Figure 17. Finite State Machine tool - Configuration tab



In the bottom part of the Configuration tab, the user can manage the device configuration using dedicated buttons:

- Read FSM Configuration: it is used to read the FSM registers and to graphically build the UI based on current FSM configuration and programs;
- Write FSM Configuration: it is used to write the entire FSM configuration (it includes FSM ODR, long counter parameters, interrupt status and programs);
- Reset All: it is used to reset the entire Finite State Machine tool UI;
- Load Device Configuration: it is used to load a .ucf file;
- Save Device Configuration: it is used to generate a .ucf file which contains both sensor and FSM register configurations.

11.1.1.1 *SM$_x$ Status*

The SMx Status groupbox is available in the top-right corner of the Configuration tab.

Figure 18. Configuration tab - SM$_x$ Status

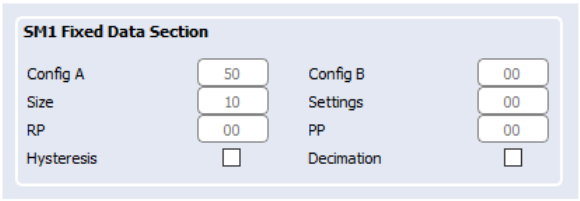
The SM$_x$ Status groupbox allows the user to enable/disable the state machine and to route the interrupt status on the INT1/INT2 pin. In detail:

- 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 the routing of the state machine interrupt on INT1 pin. This is effective if the INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to ‘1’;
- the “INT2” checkbox is used to enable the routing of the state machine interrupt on the INT2 pin. This is effective if the INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to ‘1’.

11.1.1.2 *SM$_x$ Fixed Data Section*

The SM$_x$ Fixed Data Section groupbox is available in the right part of the Configuration tab.

Figure 19. Configuration tab - SM$_x$ Fixed Data Section

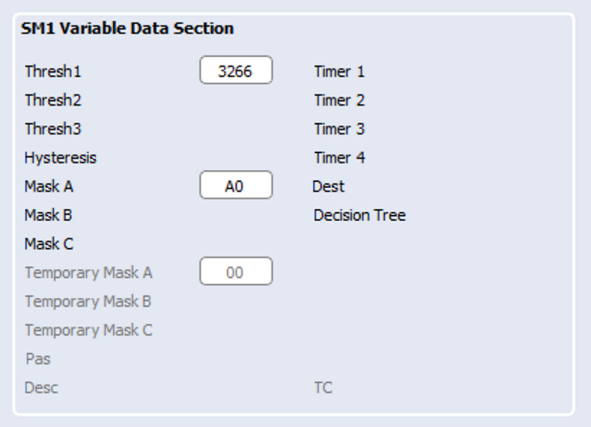


The SM$_x$ Fixed Data Section groupbox allows the user to have information about the fixed data section bytes of the program. These bytes are automatically managed by the Finite State Machine tool. It is also possible to enable/disable hysteresis and the decimation resources depending on user needs. If enabled, the corresponding resource will be shown in the SM$_x$ Variable Data Section groupbox.

11.1.1.3 *SM_x Variable Data Section*

The SM_x Variable Data Section groupbox is available in the bottom-right corner of the configuration tab.

Figure 20. Configuration tab – SM_x Variable Data Section

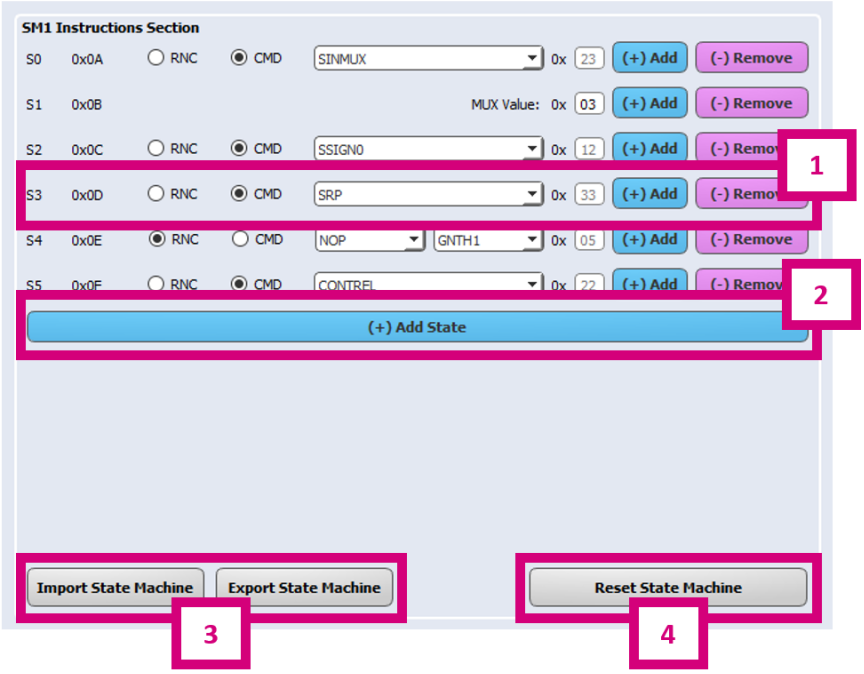


The SM_x Variable Data Section groupbox simplifies the resource allocation process: all the needed resources are automatically shown or hidden in the SM_x Variable Data Section groupbox depending on the instructions that compose the SM_x Instruction Section. The user has just to set the values of the shown resources.

11.1.1.4 *SM_x Instructions Section*

The SM_x Instructions Section groupbox is available in the left part of the Configuration tab.

Figure 21. Configuration tab – SM_x Instructions Section



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

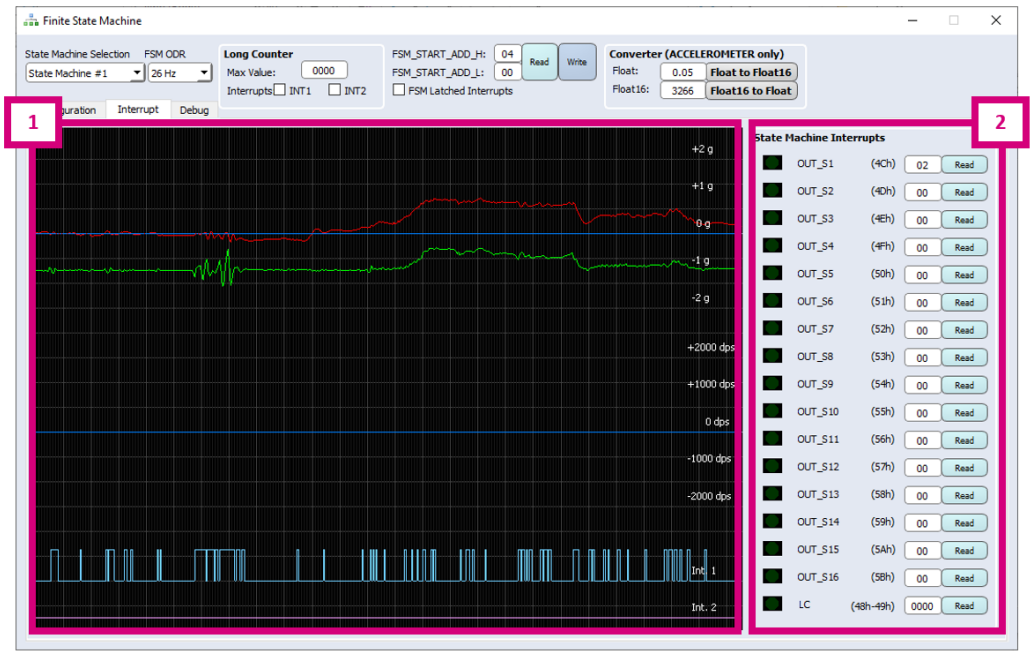
1. Customize an existing state. The single state is composed of:
   - state number S_x
   - state program relative hexadecimal address (address 0x00 corresponds to CONFIG_A byte in the fixed data section)
   - state type and opcode: user can customize the state using radio buttons and drop-down lists as described below:
     - “RNC” radio button: the state is a RESET/NEXT condition. In this case, two 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” radio button: the state is a Command. In this case, one 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.
   - “Add” button is used to insert a new state just before the current one;
   - “Remove” button is used to remove the current state.
2. “Add State” button is used to 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. “Import / Export State Machine” buttons are used to import / export the state machine program in .fsm format. The format .fsm is used to allow the user to build the entire FSM configuration starting from a set of .fsm state machine programs.
4. “Reset State Machine” button is used to reset the state machine instructions section (only on UI, not in the device).

11.1.2 Interrupt tab

The Interrupt tab of the Finite State Machine tool allows the user to check the functionality of the configured programs at runtime of the program logic. The UI is composed of two parts as shown in Figure 22.

1. Signal plots: a plot of the accelerometer and interrupt signals is shown here based on the enabled sensor and interrupt configuration;
2. State Machine Interrupt status: in this groupbox, two columns of information are shown:
   – a graphic green 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 ~300 msec;
   – the OUT_Sx register value and the long counter register value can be manually read by clicking on the corresponding "Read" button.

Figure 22. Finite State Machine tool - Interrupt tab

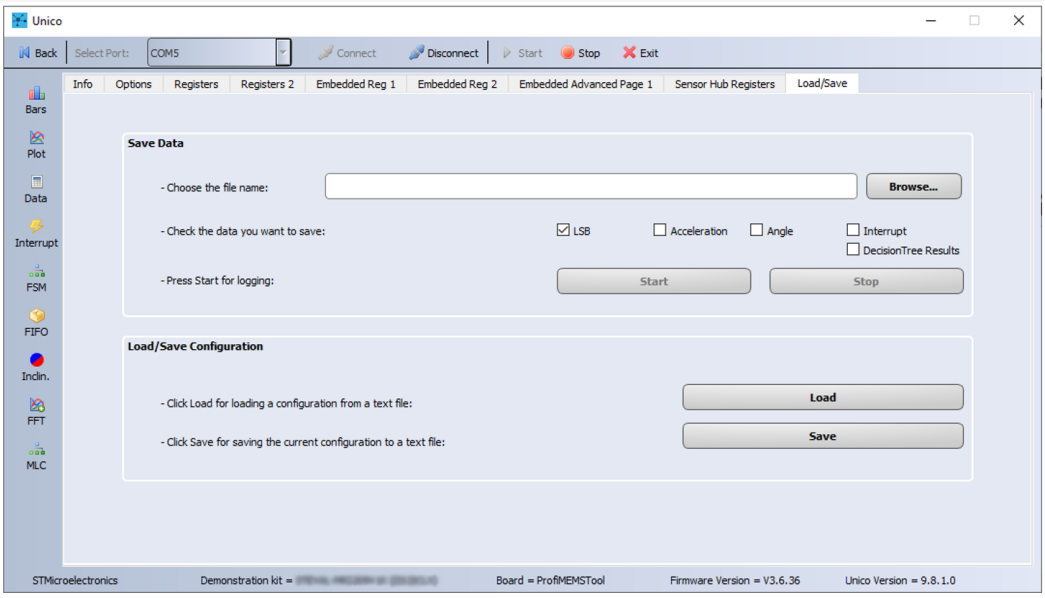


11.1.3 Debug tab

The debug tab can be used to inject data into the device in order to check the functionality of the configured programs.

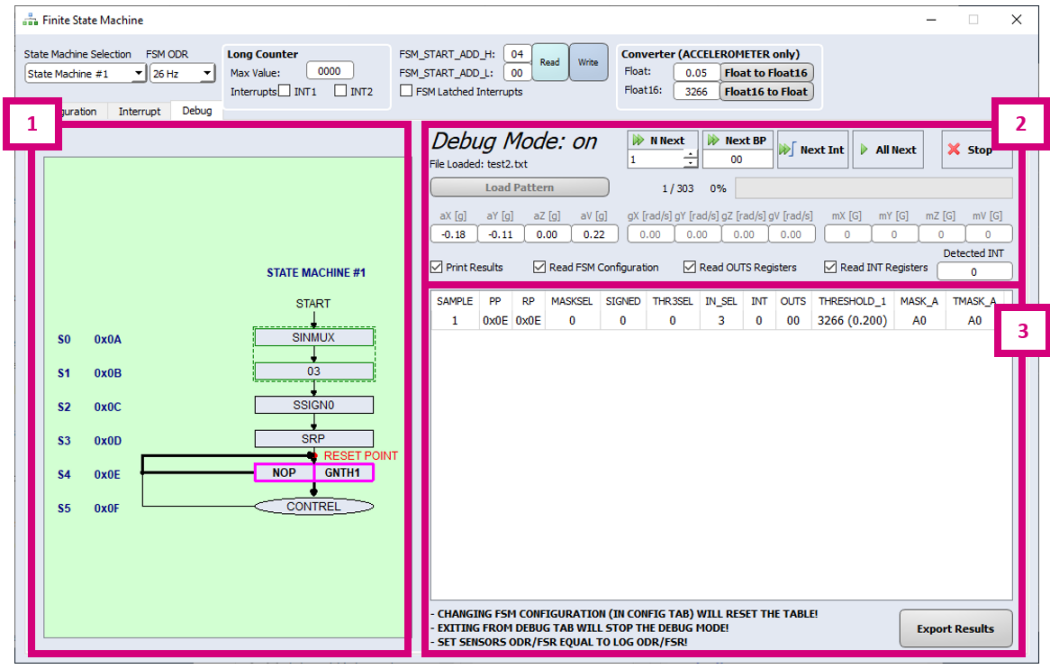
The UNICO GUI Load/Save tab, shown in the following figure, allows the user to take log files properly formatted for the data injection procedure: these log files have to contain [LSB] data only.

Figure 23. UNICO GUI – Load/Save tab



The debug tab window is shown in the following figure.

Figure 24. Finite State Machine tool – Debug tab



The debug tab is mainly composed of three UI parts:

1. State machines 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.
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. Injected sample data and the number of detected interrupts are shown here.
3. Output results: after injecting a sample into the device, a new line is added to the table depending on the “Print Results” checkbox status. Table columns represent the state machine parameters and resources, while table rows are related to the injected sample. When a parameter or a resource value is changed, the corresponding cell is highlighted. Finally, it is possible to export the table results in a text file format.

Revision history

Table 34. **Document revision history**

Date	Version	Changes
04-Aug-2020	1	Initial release
16-Oct-2020	2	Updated Section 2 Signal Conditioning block Updated Table 5. Embedded function registers and Section 3.1.2.16 FSM_OUTS[1:16] (4Ch - 5Bh) Updated Section 3.2.1 Input Selector block Updated Section 6.3 Masks / temporary masks Updated Section 7.1 Reset/Next conditions Updated TI1 (1h) through CHKDT (Fh) Updated SINMUX (23h) Updated Figure 10. Variable Data section Updated Section 8 FSM configuration example Removed section regarding "Wakeup" Updated Figure 17, Figure 19, Figure 20, Figure 21, Figure 22, Figure 24
25-Jan-2021	3	Updated Figure 6. FSM inputs

Contents

List of tables

List of figures