ISM330IS: always-on 3-axis accelerometer and 3-axis gyroscope with ISPU - intelligent sensor processing unit

Introduction

This document provides usage information and application hints related to ST's ISM330IS iNEMO inertial module with intelligent sensor processing unit (ISPU).

The ISM330IS is a 3-axis digital accelerometer and 3-axis digital gyroscope system-in-package with a digital I²C/SPI serial interface standard output, performing at 0.59 mA in combo high-performance mode (gyroscope + accelerometer only, ISPU not included). Thanks to the ultralow noise performance of both the gyroscope and the accelerometer, the device combines always-on low-power features with superior sensing precision for an optimal motion experience for the consumer.

The device has a dynamic user-selectable full-scale acceleration range of ±2/±4/±8/±16 *g* and an angular rate range of ±125/±250/±500/±1000/±2000 dps.

The availability of a dedicated connection mode with up to 4 external sensors allows the implementation of the sensor hub functionality.

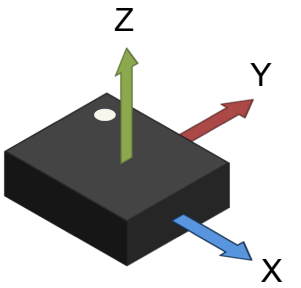
The ISM330IS embeds a new ST category of processing, ISPU - intelligent sensor processing unit, to support real-time applications that rely on sensor data. The ISPU is an ultralow-power, high-performance programmable core based on the STRED architecture, a proprietary architecture developed by STMicroelectronics. The ISPU toolchain allows developing in C code and loading any custom program in the core, ranging from signal processing algorithms to machine learning and deep learning models.

The ISM330IS is available in a small plastic land grid array package (LGA-14L) and it is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

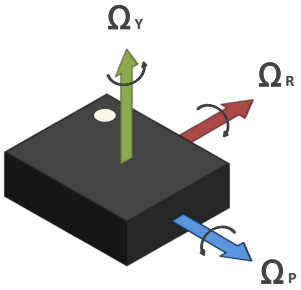
The ultrasmall size and weight of the SMD package make it an ideal choice for industrial applications such as robotics, anomaly detection, and asset tracking.

1 Pin description

Figure 1. Pin connections



Direction of detectable acceleration (top view)



Direction of detectable angular rate (top view)

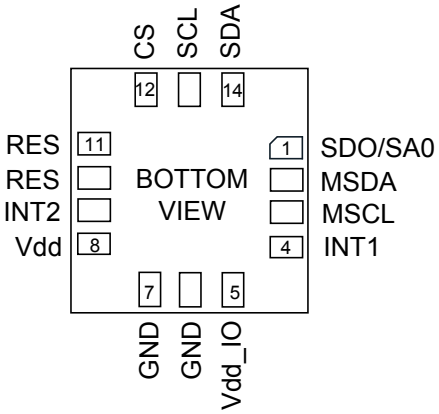


Table 1. Internal pin status

Pin #	Name	Mode 1 function	Mode 2 function	Pin status - mode 1	Pin status - mode 2
1	SDO	SPI 4-wire interface serial data output (SDO)	SPI 4-wire interface serial data output (SDO)	Default: input without pull-up Pull-up is enabled if bit SDO_PU_EN = 1 in PIN_CTRL (02h).	Default: input without pull-up Pull-up is enabled if bit SDO_PU_EN = 1 in PIN_CTRL (02h).
	SA0	I²C least significant bit of the device address (SA0)	I²C least significant bit of the device address (SA0)		
2	MSDA	Connect to Vdd_IO or GND	I²C serial data master (MSDA)	Default: input without pull-up Pull-up is enabled if bit SHUB_PU_EN = 1 in MASTER_CONFIG (14h) in the sensor hub registers (see Note to enable pull-up).	Default: input without pull-up Pull-up is enabled if bit SHUB_PU_EN = 1 in MASTER_CONFIG (14h) in the sensor hub registers (see Note to enable pull-up).
3	MSCL	Connect to Vdd_IO or GND	I²C serial clock master (MSCL)	Default: input without pull-up Pull-up is enabled if bit SHUB_PU_EN = 1 in MASTER_CONFIG (14h) in the sensor hub registers (see Note to enable pull-up).	Default: input without pull-up Pull-up is enabled if bit SHUB_PU_EN = 1 in MASTER_CONFIG (14h) in the sensor hub registers (see Note to enable pull-up).
4	INT1	Programmable interrupt 1	Programmable interrupt 1	Default: input with pull-down(1)	Default: input with pull-down(1)
5	Vdd_IO	Power supply for I/O pins	Power supply for I/O pins		
6	GND	0 V supply	0 V supply		
7	GND	0 V supply	0 V supply		
8	Vdd	Power supply	Power supply		
9	INT2	Programmable interrupt 2 (INT2)	Programmable interrupt 2 (INT2) / I²C master external synchronization signal (MDRDY)	Default: output forced to ground	Default: output forced to ground
10	RES	Leave unconnected	Leave unconnected		
11	RES	Connect to Vdd_IO or leave unconnected	Connect to Vdd_IO or leave unconnected		
12	CS	I²C / SPI mode selection (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)	I²C / SPI mode selection (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)	Default: input with pull-up	Default: input with pull-up
13	SCL	I²C serial clock (SCL) / SPI serial port clock (SPC)	I²C serial clock (SCL) / SPI serial port clock (SPC)	Default: input without pull-up	Default: input without pull-up
14	SDA	I²C serial data (SDA) / SPI serial data input (SDI) / 3-wire interface serial data output (SDO)	I²C serial data (SDA) / SPI serial data input (SDI) / 3-wire interface serial data output (SDO)	Default: input without pull-up	Default: input without pull-up

1. *INT1 must be set to 0 or left unconnected during power-on.*

Internal pull-up value is from 30 kΩ to 50 kΩ, depending on Vdd_IO.

Note: The procedure to enable the pull-up on pins 2 and 3 is as follows:

1. From the primary I²C/SPI interface: write 40h in register at address 01h (enable access to the sensor hub registers)

2. From the primary I²C/SPI interface: write 08h in register at address 14h (enable the pull-up on pins 2 and 3)

3. From the primary I²C/SPI interface: write 00h in register at address 01h (disable access to the sensor hub registers)

2 Registers

Table 2. Registers

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FUNC_CFG_ACCESS	01h	ISPU_REG _ACCESS	SHUB_REG _ACCESS	0	0	0	0	SW_RESET _ISPU	0
PIN_CTRL	02h	0	SDO_PU_EN	1	1	1	1	1	1
DRDY_PULSED_REG	0Bh	DRDY_PULSED	0	0	0	0	0	0	0
INT1_CTRL	0Dh	0	0	0	0	0	INT1_BOOT	INT1_DRDY_G	INT1_DRDY_XL
INT2_CTRL	0Eh	INT2_SLEEP _ISPU	0	0	0	0	INT2_DRDY_TEMP	INT2_DRDY_G	INT2_DRDY_XL
WHO_AM_I	0Fh	0	0	1	0	0	0	1	0
CTRL1_XL	10h	ODR_XL3	ODR_XL2	ODR_XL1	ODR_XL0	FS1_XL	FS0_XL	0	0
CTRL2_G	11h	ODR_G3	ODR_G2	ODR_G1	ODR_G0	FS1_G	FS0_G	FS_125	0
CTRL3_C	12h	BOOT	BDU	H_LACTIVE	PP_OD	SIM	IF_INC	0	SW_RESET
CTRL4_C	13h	0	SLEEP_G	INT2_on_INT1	0	0	I2C_disable	0	0
CTRL5_C	14h	0	0	0	0	ST1_G	ST0_G	ST1_XL	ST0_XL
CTRL6_C	15h	0	0	0	XL_HM_MODE	0	0	0	0
CTRL7_G	16h	G_HM_MODE	0	0	0	0	0	0	0
CTRL9_C	18h	ISPU_RATE_3	ISPU_RATE_2	ISPU_RATE_1	ISPU_RATE_0	0	0	ISPU_BDU_1	ISPU_BDU_0
CTRL10_C	19h	0	0	TIMESTAMP_EN	0	0	ISPU_CLK_SEL	0	0
ISPU_INT_STATUS0_MAINPAGE	1Ah	IA_ISPU_7	IA_ISPU_6	IA_ISPU_5	IA_ISPU_4	IA_ISPU_3	IA_ISPU_2	IA_ISPU_1	IA_ISPU_0
ISPU_INT_STATUS1_MAINPAGE	1Bh	IA_ISPU_15	IA_ISPU_14	IA_ISPU_13	IA_ISPU_12	IA_ISPU_11	IA_ISPU_10	IA_ISPU_9	IA_ISPU_8
ISPU_INT_STATUS2_MAINPAGE	1Ch	IA_ISPU_23	IA_ISPU_22	IA_ISPU_21	IA_ISPU_20	IA_ISPU_19	IA_ISPU_18	IA_ISPU_17	IA_ISPU_16
ISPU_INT_STATUS3_MAINPAGE	1Dh	0	0	IA_ISPU_29	IA_ISPU_28	IA_ISPU_27	IA_ISPU_26	IA_ISPU_25	IA_ISPU_24
STATUS_REG	1Eh	TIMESTAMP_ ENDCOUNT	0	0	0	0	TDA	GDA	XLDA
OUT_TEMP_L	20h	Temp7	Temp6	Temp5	Temp4	Temp3	Temp2	Temp1	Temp0
OUT_TEMP_H	21h	Temp15	Temp14	Temp13	Temp12	Temp11	Temp10	Temp9	Temp8
OUTX_L_G	22h	D7	D6	D5	D4	D3	D2	D1	D0
OUTX_H_G	23h	D15	D14	D13	D12	D11	D10	D9	D8
OUTY_L_G	24h	D7	D6	D5	D4	D3	D2	D1	D0
OUTY_H_G	25h	D15	D14	D13	D12	D11	D10	D9	D8
OUTZ_L_G	26h	D7	D6	D5	D4	D3	D2	D1	D0
OUTZ_H_G	27h	D15	D14	D13	D12	D11	D10	D9	D8
OUTX_L_A	28h	D7	D6	D5	D4	D3	D2	D1	D0
OUTX_H_A	29h	D15	D14	D13	D12	D11	D10	D9	D8

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OUTY_L_A	2Ah	D7	D6	D5	D4	D3	D2	D1	D0
OUTY_H_A	2Bh	D15	D14	D13	D12	D11	D10	D9	D8
OUTZ_L_A	2Ch	D7	D6	D5	D4	D3	D2	D1	D0
OUTZ_H_A	2Dh	D15	D14	D13	D12	D11	D10	D9	D8
STATUS_MASTER_ MAINPAGE	39h	WR_ONCE_DONE	SLAVE3_NACK	SLAVE2_NACK	SLAVE1_NACK	SLAVE0_NACK	0	0	SENS_HUB _ENDOP
TIMESTAMP0	40h	D7	D6	D5	D4	D3	D2	D1	D0
TIMESTAMP1	41h	D15	D14	D13	D12	D11	D10	D9	D8
TIMESTAMP2	42h	D23	D22	D21	D20	D19	D18	D17	D16
TIMESTAMP3	43h	D31	D30	D29	D28	D27	D26	D25	D24
MD1_CFG	5Eh	0	0	0	0	0	0	INT1_ISPU	INT1_SHUB
MD2_CFG	5Fh	0	0	0	0	0	0	INT2_ISPU	INT2_TIMESTAMP
INTERNAL_FREQ_FINE	63h	FREQ_FINE7	FREQ_FINE6	FREQ_FINE5	FREQ_FINE4	FREQ_FINE3	FREQ_FINE2	FREQ_FINE1	FREQ_FINE0
ISPU_DUMMY_CFG_1_L	73h	ISPU_DUMMY _CFG_1_7	ISPU_DUMMY _CFG_1_6	ISPU_DUMMY _CFG_1_5	ISPU_DUMMY _CFG_1_4	ISPU_DUMMY _CFG_1_3	ISPU_DUMMY _CFG_1_2	ISPU_DUMMY _CFG_1_1	ISPU_DUMMY _CFG_1_0
ISPU_DUMMY_CFG_1_H	74h	ISPU_DUMMY _CFG_1_15	ISPU_DUMMY _CFG_1_14	ISPU_DUMMY _CFG_1_13	ISPU_DUMMY _CFG_1_12	ISPU_DUMMY _CFG_1_11	ISPU_DUMMY _CFG_1_10	ISPU_DUMMY _CFG_1_9	ISPU_DUMMY _CFG_1_8
ISPU_DUMMY_CFG_2_L	75h	ISPU_DUMMY _CFG_2_7	ISPU_DUMMY _CFG_2_6	ISPU_DUMMY _CFG_2_5	ISPU_DUMMY _CFG_2_4	ISPU_DUMMY _CFG_2_3	ISPU_DUMMY _CFG_2_2	ISPU_DUMMY _CFG_2_1	ISPU_DUMMY _CFG_2_0
ISPU_DUMMY_CFG_2_H	76h	ISPU_DUMMY _CFG_2_15	ISPU_DUMMY _CFG_2_14	ISPU_DUMMY _CFG_2_13	ISPU_DUMMY _CFG_2_12	ISPU_DUMMY _CFG_2_11	ISPU_DUMMY _CFG_2_10	ISPU_DUMMY _CFG_2_9	ISPU_DUMMY _CFG_2_8
ISPU_DUMMY_CFG_3_L	77h	ISPU_DUMMY _CFG_3_7	ISPU_DUMMY _CFG_3_6	ISPU_DUMMY _CFG_3_5	ISPU_DUMMY _CFG_3_4	ISPU_DUMMY _CFG_3_3	ISPU_DUMMY _CFG_3_2	ISPU_DUMMY _CFG_3_1	ISPU_DUMMY _CFG_3_0
ISPU_DUMMY_CFG_3_H	78h	ISPU_DUMMY _CFG_3_15	ISPU_DUMMY _CFG_3_14	ISPU_DUMMY _CFG_3_13	ISPU_DUMMY _CFG_3_12	ISPU_DUMMY _CFG_3_11	ISPU_DUMMY _CFG_3_10	ISPU_DUMMY _CFG_3_9	ISPU_DUMMY _CFG_3_8
ISPU_DUMMY_CFG_4_L	79h	ISPU_DUMMY _CFG_4_7	ISPU_DUMMY _CFG_4_6	ISPU_DUMMY _CFG_4_5	ISPU_DUMMY _CFG_4_4	ISPU_DUMMY _CFG_4_3	ISPU_DUMMY _CFG_4_2	ISPU_DUMMY _CFG_4_1	ISPU_DUMMY _CFG_4_0
ISPU_DUMMY_CFG_4_H	7Ah	ISPU_DUMMY _CFG_4_15	ISPU_DUMMY _CFG_4_14	ISPU_DUMMY _CFG_4_13	ISPU_DUMMY _CFG_4_12	ISPU_DUMMY _CFG_4_11	ISPU_DUMMY _CFG_4_10	ISPU_DUMMY _CFG_4_9	ISPU_DUMMY _CFG_4_8

2.1 ISPU interaction registers

The list of the registers for the ISPU functions available in the device is given in Table 3. ISPU interaction registers and Table 4. ISPU to external resources. The ISPU interaction registers are accessible over the I²C/SPI interface when the ISPU_REG_ACCESS bit is set to 1 in the FUNC_CFG_ACCESS (01h) register. These registers are also accessible from ISPU through the address indicated in the third column (ISPU address), regardless of the configuration of the ISPU_REG_ACCESS bit.

Table 3. ISPU interaction registers

Register name	Address	ISPU address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ISPU_CONFIG	02h	6802h	0	0	0	LATCHED	0	0	CLK_DIS	ISPU_RST_N
ISPU_STATUS	04h	6804h	0	0	0	0	0	BOOT_END	0	0
ISPU_MEM_SEL	08h	-	0	READ_MEM_EN	0	0	0	0	0	MEM_SEL
ISPU_MEM_ADDR1	09h	-	MEM_ADDR_15	MEM_ADDR_14	MEM_ADDR_13	MEM_ADDR_12	MEM_ADDR_11	MEM_ADDR_10	MEM_ADDR_9	MEM_ADDR_8
ISPU_MEM_ADDR0	0Ah	-	MEM_ADDR_7	MEM_ADDR_6	MEM_ADDR_5	MEM_ADDR_4	MEM_ADDR_3	MEM_ADDR_2	MEM_ADDR_1	MEM_ADDR_0
ISPU_MEM_DATA	0Bh	-	MEM_DATA_7	MEM_DATA_6	MEM_DATA_5	MEM_DATA_4	MEM_DATA_3	MEM_DATA_2	MEM_DATA_1	MEM_DATA_0
ISPU_IF2S_FLAG_L	0Ch	680Ch	IF2S_7	IF2S_6	IF2S_5	IF2S_4	IF2S_3	IF2S_2	IF2S_1	IF2S_0
ISPU_IF2S_FLAG_H	0Dh	680Dh	IF2S_15	IF2S_14	IF2S_13	IF2S_12	IF2S_11	IF2S_10	IF2S_9	IF2S_8
ISPU_S2IF_FLAG_L	0Eh	680Eh	S2IF_7	S2IF_6	S2IF_5	S2IF_4	S2IF_3	S2IF_2	S2IF_1	S2IF_0
ISPU_S2IF_FLAG_H	0Fh	680Fh	S2IF_15	S2IF_14	S2IF_13	S2IF_12	S2IF_11	S2IF_10	S2IF_9	S2IF_8
ISPU_DOUT_00_L	10h	6810h	DOUT0_7	DOUT0_6	DOUT0_5	DOUT0_4	DOUT0_3	DOUT0_2	DOUT0_1	DOUT0_0
ISPU_DOUT_00_H	11h	6811h	DOUT0_15	DOUT0_14	DOUT0_13	DOUT0_12	DOUT0_11	DOUT0_10	DOUT0_9	DOUT0_8
ISPU_DOUT_01_L	12h	6812h	DOUT1_7	DOUT1_6	DOUT1_5	DOUT1_4	DOUT1_3	DOUT1_2	DOUT1_1	DOUT1_0
ISPU_DOUT_01_H	13h	6813h	DOUT1_15	DOUT1_14	DOUT1_13	DOUT1_12	DOUT1_11	DOUT1_10	DOUT1_9	DOUT1_8
ISPU_DOUT_02_L	14h	6814h	DOUT2_7	DOUT2_6	DOUT2_5	DOUT2_4	DOUT2_3	DOUT2_2	DOUT2_1	DOUT2_0
ISPU_DOUT_02_H	15h	6815h	DOUT2_15	DOUT2_14	DOUT2_13	DOUT2_12	DOUT2_11	DOUT2_10	DOUT2_9	DOUT2_8
ISPU_DOUT_03_L	16h	6816h	DOUT3_7	DOUT3_6	DOUT3_5	DOUT3_4	DOUT3_3	DOUT3_2	DOUT3_1	DOUT3_0
ISPU_DOUT_03_H	17h	6817h	DOUT3_15	DOUT3_14	DOUT3_13	DOUT3_12	DOUT3_11	DOUT3_10	DOUT3_9	DOUT3_8
ISPU_DOUT_04_L	18h	6818h	DOUT4_7	DOUT4_6	DOUT4_5	DOUT4_4	DOUT4_3	DOUT4_2	DOUT4_1	DOUT4_0
ISPU_DOUT_04_H	19h	6819h	DOUT4_15	DOUT4_14	DOUT4_13	DOUT4_12	DOUT4_11	DOUT4_10	DOUT4_9	DOUT4_8
ISPU_DOUT_05_L	1Ah	681Ah	DOUT5_7	DOUT5_6	DOUT5_5	DOUT5_4	DOUT5_3	DOUT5_2	DOUT5_1	DOUT5_0
ISPU_DOUT_05_H	1Bh	681Bh	DOUT5_15	DOUT5_14	DOUT5_13	DOUT5_12	DOUT5_11	DOUT5_10	DOUT5_9	DOUT5_8
ISPU_DOUT_06_L	1Ch	681Ch	DOUT6_7	DOUT6_6	DOUT6_5	DOUT6_4	DOUT6_3	DOUT6_2	DOUT6_1	DOUT6_0
ISPU_DOUT_06_H	1Dh	681Dh	DOUT6_15	DOUT6_14	DOUT6_13	DOUT6_12	DOUT6_11	DOUT6_10	DOUT6_9	DOUT6_8
ISPU_DOUT_07_L	1Eh	681Eh	DOUT7_7	DOUT7_6	DOUT7_5	DOUT7_4	DOUT7_3	DOUT7_2	DOUT7_1	DOUT7_0
ISPU_DOUT_07_H	1Fh	681Fh	DOUT7_15	DOUT7_14	DOUT7_13	DOUT7_12	DOUT7_11	DOUT7_10	DOUT7_9	DOUT7_8
ISPU_DOUT_08_L	20h	6820h	DOUT8_7	DOUT8_6	DOUT8_5	DOUT8_4	DOUT8_3	DOUT8_2	DOUT8_1	DOUT8_0
ISPU_DOUT_08_H	21h	6821h	DOUT8_15	DOUT8_14	DOUT8_13	DOUT8_12	DOUT8_11	DOUT8_10	DOUT8_9	DOUT8_8

Register name	Address	ISPU address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ISPU_DOUT_09_L	22h	6822h	DOUT9_7	DOUT9_6	DOUT9_5	DOUT9_4	DOUT9_3	DOUT9_2	DOUT9_1	DOUT9_0
ISPU_DOUT_09_H	23h	6823h	DOUT9_15	DOUT9_14	DOUT9_13	DOUT9_12	DOUT9_11	DOUT9_10	DOUT9_9	DOUT9_8
ISPU_DOUT_10_L	24h	6824h	DOUT10_7	DOUT10_6	DOUT10_5	DOUT10_4	DOUT10_3	DOUT10_2	DOUT10_1	DOUT10_0
ISPU_DOUT_10_H	25h	6825h	DOUT10_15	DOUT10_14	DOUT10_13	DOUT10_12	DOUT10_11	DOUT10_10	DOUT10_9	DOUT10_8
ISPU_DOUT_11_L	26h	6826h	DOUT11_7	DOUT11_6	DOUT11_5	DOUT11_4	DOUT11_3	DOUT11_2	DOUT11_1	DOUT11_0
ISPU_DOUT_11_H	27h	6827h	DOUT11_15	DOUT11_14	DOUT11_13	DOUT11_12	DOUT11_11	DOUT11_10	DOUT11_9	DOUT11_8
ISPU_DOUT_12_L	28h	6828h	DOUT12_7	DOUT12_6	DOUT12_5	DOUT12_4	DOUT12_3	DOUT12_2	DOUT12_1	DOUT12_0
ISPU_DOUT_12_H	29h	6829h	DOUT12_15	DOUT12_14	DOUT12_13	DOUT12_12	DOUT0_11	DOUT12_10	DOUT12_9	DOUT12_8
ISPU_DOUT_13_L	2Ah	682Ah	DOUT13_7	DOUT13_6	DOUT13_5	DOUT13_4	DOUT13_3	DOUT13_2	DOUT13_1	DOUT13_0
ISPU_DOUT_13_H	2Bh	682Bh	DOUT13_15	DOUT13_14	DOUT13_13	DOUT13_12	DOUT13_11	DOUT13_10	DOUT13_9	DOUT13_8
ISPU_DOUT_14_L	2Ch	682Ch	DOUT14_7	DOUT14_6	DOUT14_5	DOUT14_4	DOUT14_3	DOUT14_2	DOUT14_1	DOUT14_0
ISPU_DOUT_14_H	2Dh	682Dh	DOUT14_15	DOUT14_14	DOUT14_13	DOUT14_12	DOUT14_11	DOUT14_10	DOUT14_9	DOUT14_8
ISPU_DOUT_15_L	2Eh	682Eh	DOUT15_7	DOUT15_6	DOUT15_5	DOUT15_4	DOUT15_3	DOUT15_2	DOUT15_1	DOUT15_0
ISPU_DOUT_15_H	2Fh	682Fh	DOUT15_15	DOUT15_14	DOUT15_13	DOUT15_12	DOUT15_11	DOUT15_10	DOUT15_9	DOUT15_8
ISPU_DOUT_16_L	30h	6830h	DOUT16_7	DOUT16_6	DOUT16_5	DOUT16_4	DOUT16_3	DOUT16_2	DOUT16_1	DOUT16_0
ISPU_DOUT_16_H	31h	6831h	DOUT16_15	DOUT16_14	DOUT16_13	DOUT16_12	DOUT16_11	DOUT16_10	DOUT16_9	DOUT16_8
ISPU_DOUT_17_L	32h	6832h	DOUT17_7	DOUT17_6	DOUT17_5	DOUT17_4	DOUT17_3	DOUT17_2	DOUT17_1	DOUT17_0
ISPU_DOUT_17_H	33h	6833h	DOUT17_15	DOUT17_14	DOUT17_13	DOUT17_12	DOUT17_11	DOUT17_10	DOUT17_9	DOUT17_8
ISPU_DOUT_18_L	34h	6834h	DOUT18_7	DOUT18_6	DOUT18_5	DOUT18_4	DOUT18_3	DOUT18_2	DOUT18_1	DOUT18_0
ISPU_DOUT_18_H	35h	6835h	DOUT18_15	DOUT18_14	DOUT18_13	DOUT18_12	DOUT18_11	DOUT18_10	DOUT18_9	DOUT18_8
ISPU_DOUT_19_L	36h	6836h	DOUT19_7	DOUT19_6	DOUT19_5	DOUT19_4	DOUT19_3	DOUT19_2	DOUT19_1	DOUT19_0
ISPU_DOUT_19_H	37h	6837h	DOUT19_15	DOUT19_14	DOUT19_13	DOUT19_12	DOUT19_11	DOUT19_10	DOUT19_9	DOUT19_8
ISPU_DOUT_20_L	38h	6838h	DOUT20_7	DOUT20_6	DOUT20_5	DOUT20_4	DOUT20_3	DOUT20_2	DOUT20_1	DOUT20_0
ISPU_DOUT_20_H	39h	6839h	DOUT20_15	DOUT20_14	DOUT20_13	DOUT20_12	DOUT20_11	DOUT20_10	DOUT20_9	DOUT20_8
ISPU_DOUT_21_L	3Ah	683Ah	DOUT21_7	DOUT21_6	DOUT21_5	DOUT21_4	DOUT21_3	DOUT21_2	DOUT21_1	DOUT21_0
ISPU_DOUT_21_H	3Bh	683Bh	DOUT21_15	DOUT21_14	DOUT21_13	DOUT21_12	DOUT21_11	DOUT21_10	DOUT21_9	DOUT21_8
ISPU_DOUT_22_L	3Ch	683Ch	DOUT22_7	DOUT22_6	DOUT22_5	DOUT22_4	DOUT22_3	DOUT22_2	DOUT22_1	DOUT22_0
ISPU_DOUT_22_H	3Dh	683Dh	DOUT22_15	DOUT22_14	DOUT22_13	DOUT22_12	DOUT22_11	DOUT22_10	DOUT22_9	DOUT22_8
ISPU_DOUT_23_L	3Eh	683Eh	DOUT23_7	DOUT23_6	DOUT23_5	DOUT23_4	DOUT23_3	DOUT23_2	DOUT23_1	DOUT23_0
ISPU_DOUT_23_H	3Fh	683Fh	DOUT23_15	DOUT23_14	DOUT23_13	DOUT23_12	DOUT23_11	DOUT23_10	DOUT23_9	DOUT23_8
ISPU_DOUT_24_L	40h	6840h	DOUT24_7	DOUT24_6	DOUT24_5	DOUT24_4	DOUT24_3	DOUT24_2	DOUT24_1	DOUT24_0
ISPU_DOUT_24_H	41h	6841h	DOUT24_15	DOUT24_14	DOUT24_13	DOUT24_12	DOUT24_11	DOUT24_10	DOUT24_9	DOUT24_8
ISPU_DOUT_25_L	42h	6842h	DOUT25_7	DOUT25_6	DOUT25_5	DOUT25_4	DOUT25_3	DOUT25_2	DOUT25_1	DOUT25_0
ISPU_DOUT_25_H	43h	6843h	DOUT25_15	DOUT25_14	DOUT25_13	DOUT25_12	DOUT25_11	DOUT25_10	DOUT25_9	DOUT25_8

Register name	Address	ISPU address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ISPU_DOUT_26_L	44h	6844h	DOUT26_7	DOUT26_6	DOUT26_5	DOUT26_4	DOUT26_3	DOUT26_2	DOUT26_1	DOUT26_0
ISPU_DOUT_26_H	45h	6845h	DOUT26_15	DOUT26_14	DOUT26_13	DOUT26_12	DOUT26_11	DOUT26_10	DOUT26_9	DOUT26_8
ISPU_DOUT_27_L	46h	6846h	DOUT27_7	DOUT27_6	DOUT27_5	DOUT27_4	DOUT27_3	DOUT27_2	DOUT27_1	DOUT27_0
ISPU_DOUT_27_H	47h	6847h	DOUT27_15	DOUT27_14	DOUT27_13	DOUT27_12	DOUT27_11	DOUT27_10	DOUT27_9	DOUT27_8
ISPU_DOUT_28_L	48h	6848h	DOUT28_7	DOUT28_6	DOUT28_5	DOUT28_4	DOUT28_3	DOUT28_2	DOUT28_1	DOUT28_0
ISPU_DOUT_28_H	49h	6849h	DOUT28_15	DOUT28_14	DOUT28_13	DOUT28_12	DOUT28_11	DOUT28_10	DOUT28_9	DOUT28_8
ISPU_DOUT_29_L	4Ah	684Ah	DOUT29_7	DOUT29_6	DOUT29_5	DOUT29_4	DOUT29_3	DOUT29_2	DOUT29_1	DOUT29_0
ISPU_DOUT_29_H	4Bh	684Bh	DOUT29_15	DOUT29_14	DOUT29_13	DOUT29_12	DOUT29_11	DOUT29_10	DOUT29_9	DOUT29_8
ISPU_DOUT_30_L	4Ch	684Ch	DOUT30_7	DOUT30_6	DOUT30_5	DOUT30_4	DOUT30_3	DOUT30_2	DOUT30_1	DOUT30_0
ISPU_DOUT_30_H	4Dh	684Dh	DOUT30_15	DOUT30_14	DOUT30_13	DOUT30_12	DOUT30_11	DOUT30_10	DOUT30_9	DOUT30_8
ISPU_DOUT_31_L	4Eh	684Eh	DOUT31_7	DOUT31_6	DOUT10_5	DOUT31_4	DOUT31_3	DOUT31_2	DOUT31_1	DOUT31_0
ISPU_DOUT_31_H	4Fh	684Fh	DOUT31_15	DOUT31_14	DOUT31_13	DOUT31_12	DOUT31_11	DOUT31_10	DOUT31_9	DOUT31_8
ISPU_INT1_CTRL0	50h	6850h	ISPU_INT1 _CTRL7	ISPU_INT1 _CTRL6	ISPU_INT1 _CTRL5	ISPU_INT1 _CTRL4	ISPU_INT1 _CTRL3	ISPU_INT1 _CTRL2	ISPU_INT1 _CTRL1	ISPU_INT1 _CTRL0
ISPU_INT1_CTRL1	51h	6851h	ISPU_INT1 _CTRL15	ISPU_INT1 _CTRL14	ISPU_INT1 _CTRL13	ISPU_INT1 _CTRL12	ISPU_INT1 _CTRL11	ISPU_INT1 _CTRL10	ISPU_INT1 _CTRL9	ISPU_INT1 _CTRL8
ISPU_INT1_CTRL2	52h	6852h	ISPU_INT1 _CTRL23	ISPU_INT1 _CTRL22	ISPU_INT1 _CTRL21	ISPU_INT1 _CTRL20	ISPU_INT1 _CTRL19	ISPU_INT1 _CTRL18	ISPU_INT1 _CTRL17	ISPU_INT1 _CTRL16
ISPU_INT1_CTRL3	53h	6853h	0	0	ISPU_INT1 _CTRL29	ISPU_INT1 _CTRL28	ISPU_INT1 _CTRL27	ISPU_INT1 _CTRL26	ISPU_INT1 _CTRL25	ISPU_INT1 _CTRL24
ISPU_INT2_CTRL0	54h	6854h	ISPU_INT2 _CTRL7	ISPU_INT2 _CTRL6	ISPU_INT2 _CTRL5	ISPU_INT2 _CTRL4	ISPU_INT2 _CTRL3	ISPU_INT2 _CTRL2	ISPU_INT2 _CTRL1	ISPU_INT2 _CTRL0
ISPU_INT2_CTRL1	55h	6855h	ISPU_INT2 _CTRL15	ISPU_INT2 _CTRL14	ISPU_INT2 _CTRL13	ISPU_INT2 _CTRL12	ISPU_INT2 _CTRL11	ISPU_INT2 _CTRL10	ISPU_INT2 _CTRL9	ISPU_INT2 _CTRL8
ISPU_INT2_CTRL2	56h	6856h	ISPU_INT2 _CTRL23	ISPU_INT2 _CTRL22	ISPU_INT2 _CTRL21	ISPU_INT2 _CTRL20	ISPU_INT2 _CTRL19	ISPU_INT2 _CTRL18	ISPU_INT2 _CTRL17	ISPU_INT2 _CTRL16
ISPU_INT2_CTRL3	57h	6857h	0	0	ISPU_INT2 _CTRL29	ISPU_INT2 _CTRL28	ISPU_INT2 _CTRL27	ISPU_INT2 _CTRL26	ISPU_INT2 _CTRL25	ISPU_INT2 _CTRL24
ISPU_INT_STATUS0	58h	6858h	ISPU_INT_ STATUS7	ISPU_INT_ STATUS6	ISPU_INT_ STATUS5	ISPU_INT_ STATUS4	ISPU_INT_ STATUS3	ISPU_INT_ STATUS2	ISPU_INT_ STATUS1	ISPU_INT_ STATUS0
ISPU_INT_STATUS1	59h	6859h	ISPU_INT_ STATUS15	ISPU_INT_ STATUS14	ISPU_INT_ STATUS13	ISPU_INT_ STATUS12	ISPU_INT_ STATUS11	ISPU_INT_ STATUS10	ISPU_INT_ STATUS9	ISPU_INT_ STATUS8
ISPU_INT_STATUS2	5Ah	685Ah	ISPU_INT_ STATUS23	ISPU_INT_ STATUS22	ISPU_INT_ STATUS21	ISPU_INT_ STATUS20	ISPU_INT_ STATUS19	ISPU_INT_ STATUS18	ISPU_INT_ STATUS17	ISPU_INT_ STATUS16
ISPU_INT_STATUS3	5Bh	685Bh	0	0	ISPU_INT_ STATUS29	ISPU_INT_ STATUS28	ISPU_INT_ STATUS27	ISPU_INT_ STATUS26	ISPU_INT_ STATUS25	ISPU_INT_ STATUS24
ISPU_ALGO0	70h	6870h	ISPU_ALGO7	ISPU_ALGO6	ISPU_ALGO5	ISPU_ALGO4	ISPU_ALGO3	ISPU_ALGO2	ISPU_ALGO1	ISPU_ALGO0
ISPU_ALGO1	71h	6871h	ISPU_ALGO15	ISPU_ALGO14	ISPU_ALGO13	ISPU_ALGO12	ISPU_ALGO11	ISPU_ALGO10	ISPU_ALGO9	ISPU_ALGO8
ISPU_ALGO2	72h	6872h	ISPU_ALGO23	ISPU_ALGO22	ISPU_ALGO21	ISPU_ALGO20	ISPU_ALGO19	ISPU_ALGO18	ISPU_ALGO17	ISPU_ALGO16
ISPU_ALGO3	73h	6873h	0	0	ISPU_ALGO29	ISPU_ALGO28	ISPU_ALGO27	ISPU_ALGO26	ISPU_ALGO25	ISPU_ALGO24

Table 4. **ISPU to external resources**

ISPU address (hex)	Bytes	Name
6940-6941	2	TIMESTAMP0 (40h), TIMESTAMP1 (41h)
6942-6943	2	TIMESTAMP2 (42h), TIMESTAMP3 (43h)
6974-6975	2	ISPU_DUMMY_CFG_1_L (73h), ISPU_DUMMY_CFG_1_H (74h)
6976-6977	2	ISPU_DUMMY_CFG_2_L (75h), ISPU_DUMMY_CFG_2_H (76h)
6978-6979	2	ISPU_DUMMY_CFG_3_L (77h), ISPU_DUMMY_CFG_3_H (78h)
697A-697B	2	ISPU_DUMMY_CFG_4_L (79h), ISPU_DUMMY_CFG_4_H (7Ah)

2.2 ISPU functions registers

The following table provides a list of the registers internally available in the ISPU architecture.

Table 5. ISPU interaction registers

Register name	ISPU address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ISPU_GLB_CALL_EN	6800h	-	-	-	-	-	-	-	ISPU_GLB _CALL_EN
ISPU_INT_PIN	685Ch	0	0	0	0	0	0	INT2	INT1
ISPU_ARAW_X_L	6880h	ISPU_ ARAW_X_7	ISPU_ ARAW_X_6	ISPU_ ARAW_X_5	ISPU_ ARAW_X_4	ISPU_ ARAW_X_3	ISPU_ ARAW_X_2	ISPU_ ARAW_X_1	ISPU_ ARAW_X_0
ISPU_ARAW_X_H	6881h	ISPU_ ARAW_X_15	ISPU_ ARAW_X_14	ISPU_ ARAW_X_13	ISPU_ ARAW_X_12	ISPU_ ARAW_X_11	ISPU_ ARAW_X_10	ISPU_ ARAW_X_9	ISPU_ ARAW_X_8
ISPU_ARAW_Y_L	6884h	ISPU_ ARAW_Y_7	ISPU_ ARAW_Y_6	ISPU_ ARAW_Y_5	ISPU_ ARAW_Y_4	ISPU_ ARAW_Y_3	ISPU_ ARAW_Y_2	ISPU_ ARAW_Y_1	ISPU_ ARAW_Y_0
ISPU_ARAW_Y_H	6885h	ISPU_ ARAW_Y_15	ISPU_ ARAW_Y_14	ISPU_ ARAW_Y_13	ISPU_ ARAW_Y_12	ISPU_ ARAW_Y_11	ISPU_ ARAW_Y_10	ISPU_ ARAW_Y_9	ISPU_ ARAW_Y_8
ISPU_ARAW_Z_L	6888h	ISPU_ ARAW_Z_7	ISPU_ ARAW_Z_6	ISPU_ ARAW_Z_5	ISPU_ ARAW_Z_4	ISPU_ ARAW_Z_3	ISPU_ ARAW_Z_2	ISPU_ ARAW_Z_1	ISPU_ ARAW_Z_0
ISPU_ARAW_Z_H	6889h	ISPU_ ARAW_Z_15	ISPU_ ARAW_Z_14	ISPU_ ARAW_Z_13	ISPU_ ARAW_Z_12	ISPU_ ARAW_Z_11	ISPU_ ARAW_Z_10	ISPU_ ARAW_Z_9	ISPU_ ARAW_Z_8
ISPU_GRAW_X_L	688Ch	ISPU_ GRAW_X_7	ISPU_ GRAW_X_6	ISPU_ GRAW_X_5	ISPU_ GRAW_X_4	ISPU_ GRAW_X_3	ISPU_ GRAW_X_2	ISPU_ GRAW_X_1	ISPU_ GRAW_X_0
ISPU_GRAW_X_H	688Dh	ISPU_ GRAW_X_15	ISPU_ GRAW_X_14	ISPU_ GRAW_X_13	ISPU_ GRAW_X_12	ISPU_ GRAW_X_11	ISPU_ GRAW_X_10	ISPU_ GRAW_X_9	ISPU_ GRAW_X_8
ISPU_GRAW_Y_L	6890h	ISPU_ GRAW_Y_7	ISPU_ GRAW_Y_6	ISPU_ GRAW_Y_5	ISPU_ GRAW_Y_4	ISPU_ GRAW_Y_3	ISPU_ GRAW_Y_2	ISPU_ GRAW_Y_1	ISPU_ GRAW_Y_0
ISPU_GRAW_Y_H	6891h	ISPU_ GRAW_Y_15	ISPU_ GRAW_Y_14	ISPU_ GRAW_Y_13	ISPU_ GRAW_Y_12	ISPU_ GRAW_Y_11	ISPU_ GRAW_Y_10	ISPU_ GRAW_Y_9	ISPU_ GRAW_Y_8
ISPU_GRAW_Z_L	6894h	ISPU_ GRAW_Z_7	ISPU_ GRAW_Z_6	ISPU_ GRAW_Z_5	ISPU_ GRAW_Z_4	ISPU_ GRAW_Z_3	ISPU_ GRAW_Z_2	ISPU_ GRAW_Z_1	ISPU_ GRAW_Z_0
ISPU_GRAW_Z_H	6895h	ISPU_ GRAW_Z_15	ISPU_ GRAW_Z_14	ISPU_ GRAW_Z_13	ISPU_ GRAW_Z_12	ISPU_ GRAW_Z_11	ISPU_ GRAW_Z_10	ISPU_ GRAW_Z_9	ISPU_ GRAW_z_8
ISPU_ERAW_0_L	6898h	ISPU_ ERAW_0_7	ISPU_ ERAW_0_6	ISPU_ ERAW_0_5	ISPU_ ERAW_0_4	ISPU_ ERAW_0_3	ISPU_ ERAW_0_2	ISPU_ ERAW_0_1	ISPU_ ERAW_0_0
ISPU_ERAW_0_H	6899h	ISPU_ ERAW_0_15	ISPU_ ERAW_0_14	ISPU_ ERAW_0_13	ISPU_ ERAW_0_12	ISPU_ ERAW_0_11	ISPU_ ERAW_0_10	ISPU_ ERAW_0_9	ISPU_ ERAW_0_8
ISPU_ERAW_1_L	689Ch	ISPU_ ERAW_1_7	ISPU_ ERAW_1_6	ISPU_ ERAW_1_5	ISPU_ ERAW_1_4	ISPU_ ERAW_1_3	ISPU_ ERAW_1_2	ISPU_ ERAW_1_1	ISPU_ ERAW_1_0
ISPU_ERAW_1_H	689Dh	ISPU_ ERAW_1_15	ISPU_ ERAW_1_14	ISPU_ ERAW_1_13	ISPU_ ERAW_1_12	ISPU_ ERAW_1_11	ISPU_ ERAW_1_10	ISPU_ ERAW_1_9	ISPU_ ERAW_1_8
ISPU_ERAW_2_L	68A0h	ISPU_ ERAW_2_7	ISPU_ ERAW_2_6	ISPU_ ERAW_2_5	ISPU_ ERAW_2_4	ISPU_ ERAW_2_3	ISPU_ ERAW_2_2	ISPU_ ERAW_2_1	ISPU_ ERAW_2_0
ISPU_ERAW_2_H	68A1h	ISPU_ ERAW_2_15	ISPU_ ERAW_2_14	ISPU_ ERAW_2_13	ISPU_ ERAW_2_12	ISPU_ ERAW_2_11	ISPU_ ERAW_2_10	ISPU_ ERAW_2_9	ISPU_ ERAW_2_8

Register name	ISPU address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ISPU_TEMP_L	68A4h	ISPU_ TEMP_7	ISPU_ TEMP_6	ISPU_ TEMP_5	ISPU_ TEMP_4	ISPU_ TEMP_3	ISPU_ TEMP_2	ISPU_ TEMP_1	ISPU_ TEMP_0
ISPU_TEMP_H	68A5h	ISPU_ TEMP_15	ISPU_ TEMP_14	ISPU_ TEMP_13	ISPU_ TEMP_12	ISPU_ TEMP_11	ISPU_ TEMP_10	ISPU_ TEMP_9	ISPU_ TEMP_8
ISPU_CALL_EN_0	68B8h	ISPU_CALL_ ALGO_6	ISPU_CALL_ ALGO_5	ISPU_CALL_ ALGO_4	ISPU_CALL_ ALGO_3	ISPU_CALL_ ALGO_2	ISPU_CALL_ ALGO_1	ISPU_CALL_ ALGO_0	0
ISPU_CALL_EN_1	68B9h	ISPU_CALL_ ALGO_14	ISPU_CALL_ ALGO_13	ISPU_CALL_ ALGO_12	ISPU_CALL_ ALGO_11	ISPU_CALL_ ALGO_10	ISPU_CALL_ ALGO_9	ISPU_CALL_ ALGO_8	ISPU_CALL_ ALGO_7
ISPU_CALL_EN_2	68BAh	ISPU_CALL_ ALGO_22	ISPU_CALL_ ALGO_21	ISPU_CALL_ ALGO_20	ISPU_CALL_ ALGO_19	ISPU_CALL_ ALGO_18	ISPU_CALL_ ALGO_17	ISPU_CALL_ ALGO_16	ISPU_CALL_ ALGO_15
ISPU_CALL_EN_3	68BBh	0	ISPU_CALL_ ALGO_29	ISPU_CALL_ ALGO_28	ISPU_CALL_ ALGO_27	ISPU_CALL_ ALGO_26	ISPU_CALL_ ALGO_25	ISPU_CALL_ ALGO_24	ISPU_CALL_ ALGO_23
ISPU_DTIME_0_L	6948h	ISPU_DTIME_7	ISPU_DTIME_6	ISPU_DTIME_5	ISPU_DTIME_4	ISPU_DTIME_3	ISPU_DTIME_2	ISPU_DTIME_1	ISPU_DTIME_0
ISPU_DTIME_0_H	6949h	ISPU_DTIME_15	ISPU_DTIME_14	ISPU_DTIME_13	ISPU_DTIME_12	ISPU_DTIME_11	ISPU_DTIME_10	ISPU_DTIME_9	ISPU_DTIME_8
ISPU_DTIME_1_L	694Ah	ISPU_DTIME_23	ISPU_DTIME_22	ISPU_DTIME_21	ISPU_DTIME_20	ISPU_DTIME_19	ISPU_DTIME_18	ISPU_DTIME_17	ISPU_DTIME_16
ISPU_DTIME_1_H	694Bh	ISPU_DTIME_31	ISPU_DTIME_30	ISPU_DTIME_29	ISPU_DTIME_28	ISPU_DTIME_27	ISPU_DTIME_26	ISPU_DTIME_25	ISPU_DTIME_24

2.3 Sensor hub registers

The sensor hub registers are accessible when bit SHUB_REG_ACCESS is set to 1 in the FUNC_CFG_ACCESS (01h) register.

Table 6. Sensor hub registers

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SENSOR_HUB_1	02h	SensorHub1_7	SensorHub1_6	SensorHub1_5	SensorHub1_4	SensorHub1_3	SensorHub1_2	SensorHub1_1	SensorHub1_0
SENSOR_HUB_2	03h	SensorHub2_7	SensorHub2_6	SensorHub2_5	SensorHub2_4	SensorHub2_3	SensorHub2_2	SensorHub2_1	SensorHub2_0
SENSOR_HUB_3	04h	SensorHub3_7	SensorHub3_6	SensorHub3_5	SensorHub3_4	SensorHub3_3	SensorHub3_2	SensorHub3_1	SensorHub3_0
SENSOR_HUB_4	05h	SensorHub4_7	SensorHub4_6	SensorHub4_5	SensorHub4_4	SensorHub4_3	SensorHub4_2	SensorHub4_1	SensorHub4_0
SENSOR_HUB_5	06h	SensorHub5_7	SensorHub5_6	SensorHub5_5	SensorHub5_4	SensorHub5_3	SensorHub5_2	SensorHub5_1	SensorHub5_0
SENSOR_HUB_6	07h	SensorHub6_7	SensorHub6_6	SensorHub6_5	SensorHub6_4	SensorHub6_3	SensorHub6_2	SensorHub6_1	SensorHub6_0
SENSOR_HUB_7	08h	SensorHub7_7	SensorHub7_6	SensorHub7_5	SensorHub7_4	SensorHub7_3	SensorHub7_2	SensorHub7_1	SensorHub7_0
SENSOR_HUB_8	09h	SensorHub8_7	SensorHub8_6	SensorHub8_5	SensorHub8_4	SensorHub8_3	SensorHub8_2	SensorHub8_1	SensorHub8_0
SENSOR_HUB_9	0Ah	SensorHub9_7	SensorHub9_6	SensorHub9_5	SensorHub9_4	SensorHub9_3	SensorHub9_2	SensorHub9_1	SensorHub9_0
SENSOR_HUB_10	0Bh	SensorHub10_7	SensorHub10_6	SensorHub10_5	SensorHub10_4	SensorHub10_3	SensorHub10_2	SensorHub10_1	SensorHub10_0
SENSOR_HUB_11	0Ch	SensorHub11_7	SensorHub11_6	SensorHub11_5	SensorHub11_4	SensorHub11_3	SensorHub11_2	SensorHub11_1	SensorHub11_0
SENSOR_HUB_12	0Dh	SensorHub12_7	SensorHub12_6	SensorHub12_5	SensorHub12_4	SensorHub12_3	SensorHub12_2	SensorHub12_1	SensorHub12_0
SENSOR_HUB_13	0Eh	SensorHub13_7	SensorHub13_6	SensorHub13_5	SensorHub13_4	SensorHub13_3	SensorHub13_2	SensorHub13_1	SensorHub13_0
SENSOR_HUB_14	0Fh	SensorHub14_7	SensorHub14_6	SensorHub14_5	SensorHub14_4	SensorHub14_3	SensorHub14_2	SensorHub14_1	SensorHub14_0
SENSOR_HUB_15	10h	SensorHub15_7	SensorHub15_6	SensorHub15_5	SensorHub15_4	SensorHub15_3	SensorHub15_2	SensorHub15_1	SensorHub15_0
SENSOR_HUB_16	11h	SensorHub16_7	SensorHub16_6	SensorHub16_5	SensorHub16_4	SensorHub16_3	SensorHub16_2	SensorHub16_1	SensorHub16_0
SENSOR_HUB_17	12h	SensorHub17_7	SensorHub17_6	SensorHub17_5	SensorHub17_4	SensorHub17_3	SensorHub17_2	SensorHub17_1	SensorHub17_0
SENSOR_HUB_18	13h	SensorHub18_7	SensorHub18_6	SensorHub18_5	SensorHub18_4	SensorHub18_3	SensorHub18_2	SensorHub18_1	SensorHub18_0
MASTER_CONFIG	14h	RST_MASTER _REGS	WRITE_ONCE	START_CONFIG	PASS_ THROUGH_MODE	SHUB_PU_EN	MASTER_ON	AUX_SENS_ON_1	AUX_SENS_ON_0
SLV0_ADD	15h	slave0_add6	slave0_add5	slave0_add4	slave0_add3	slave0_add2	slave0_add1	slave0_add0	rw_0
SLV0_SUBADD	16h	slave0_reg7	slave0_reg6	slave0_reg5	slave0_reg4	slave0_reg3	slave0_reg2	slave0_reg1	slave0_reg0
SLV0_CONFIG	17h	SHUB_ODR_1	SHUB_ODR_0	0	0	0	Slave0_numop2	Slave0_numop1	Slave0_numop0
SLV1_ADD	18h	Slave1_add6	Slave1_add5	Slave1_add4	Slave1_add3	Slave1_add2	Slave1_add1	Slave1_add0	r_1
SLV1_SUBADD	19h	Slave1_reg7	Slave1_reg6	Slave1_reg5	Slave1_reg4	Slave1_reg3	Slave1_reg2	Slave1_reg1	Slave1_reg0
SLV1_CONFIG	1Ah	0	0	0	0	0	Slave1_numop2	Slave1_numop1	Slave1_numop0
SLV2_ADD	1Bh	Slave2_add6	Slave2_add5	Slave2_add4	Slave2_add3	Slave2_add2	Slave2_add1	Slave2_add0	r_2
SLV2_SUBADD	1Ch	Slave2_reg7	Slave2_reg6	Slave2_reg5	Slave2_reg4	Slave2_reg3	Slave2_reg2	Slave2_reg1	Slave2_reg0
SLV2_CONFIG	1Dh	0	0	0	0	0	Slave2_numop2	Slave2_numop1	Slave2_numop0
SLV3_ADD	1Eh	Slave3_add6	Slave3_add5	Slave3_add4	Slave3_add3	Slave3_add2	Slave3_add1	Slave3_add0	r_3
SLV3_SUBADD	1Fh	Slave3_reg7	Slave3_reg6	Slave3_reg5	Slave3_reg4	Slave3_reg3	Slave3_reg2	Slave3_reg1	Slave3_reg0

Register name	Address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SLV3_CONFIG	20h	0	0	0	0	0	Slave3_numop2	Slave3_numop1	Slave3_numop0
DATAWRITE_SLV0	21h	Slave0_dataw7	Slave0_dataw6	Slave0_dataw5	Slave0_dataw4	Slave0_dataw3	Slave0_dataw2	Slave0_dataw1	Slave0_dataw0
STATUS_MASTER	22h	WR_ONCE_DONE	SLAVE3_NACK	SLAVE2_NACK	SLAVE1_NACK	SLAVE0_NACK	0	0	SENS_HUB _ENDOP

3 Operating modes

The ISM330IS provides three possible operating configurations:

- only accelerometer active and gyroscope in power-down or sleep mode
- only gyroscope active and accelerometer in power-down
- both accelerometer and gyroscope active with independent ODR

The device offers a wide Vdd voltage range from 1.71 V to 3.6 V and a Vdd_IO range from 1.62 V to 3.6 V. The power-on sequence is not restricted. The Vdd/Vdd_IO pins can be set to either power supply level or to ground level (they must not be left floating) and no specific sequence is required for powering them on.

In order to avoid potential conflicts, during the power-on sequence it is recommended to set the lines (on the host side) connected to the device IO pins floating or connected to ground, until Vdd_IO is set. After Vdd_IO is set, the lines connected to the IO pins have to be configured according to their default status described in Table 1. In order to avoid an unexpected increase in current consumption, the input pins which are not pulled-up/pulled-down must be polarized by the host.

When the Vdd power supply is applied, the device performs a 10 ms (maximum) boot procedure to load the trimming parameters. After the boot is completed, both the accelerometer and the gyroscope are automatically configured in power-down mode. To guarantee proper power-off of the device it is recommended to maintain the duration of the Vdd line to GND for at least 100 µs.

The accelerometer and the gyroscope can be independently configured in three different power modes: power-down, low-power, and high-performance mode. They are allowed to have different data rates without any limit. The gyroscope sensor can also be set in sleep mode to reduce its power consumption.

When both the accelerometer and gyroscope are on, the accelerometer is synchronized with the gyroscope, and the data rates of the two sensors are integer multiples of each other.

Referring to the ISM330IS datasheet, the output data rate (ODR_XL) bits of the CTRL1_XL register and the high-performance mode (XL_HM_MODE) bit of the CTRL6_C register are used to select the output data rate and the power mode of the accelerometer (Table 7. Accelerometer ODR and power mode selection).

Table 7. Accelerometer ODR and power mode selection

ODR_XL[3:0]	ODR when XL_HM_MODE = 1	ODR when XL_HM_MODE = 0
0000	Power-down mode	Power-down mode
1011	1.6 Hz (low-power mode only)	12.5 Hz (high-performance mode)
0001	12.5 Hz (low-power mode)	12.5 Hz (high-performance mode)
0010	26 Hz (low-power mode)	26 Hz (high-performance mode)
0011	52 Hz (low-power mode)	52 Hz (high-performance mode)
0100	104 Hz (low-power mode)	104 Hz (high-performance mode)
0101	208 Hz (low-power mode)	208 Hz (high-performance mode)
0110	416 Hz (high-performance mode)	416 Hz (high-performance mode)
0111	833 Hz (high-performance mode)	833 Hz (high-performance mode)
1000	1667 Hz (high-performance mode)	1667 Hz (high-performance mode)
1001	3333 Hz (high-performance mode)	3333 Hz (high-performance mode)
1010	6667 Hz (high-performance mode)	6667 Hz (high-performance mode)

The output data rate (ODR_G) bits of the CTRL2_G register and the high-performance mode (G_HM_MODE) bit of the CTRL7_G register are used to select the output data rate and the power mode of the gyroscope sensor (Table 8. Gyroscope ODR and power mode selection).

Table 8. Gyroscope ODR and power mode selection

ODR_G[3:0]	ODR when G_HM_MODE = 1	ODR when G_HM_MODE = 0
0000	Power-down mode	Power-down mode
0001	12.5 Hz (low-power mode)	12.5 Hz (high-performance mode)
0010	26 Hz (low-power mode)	26 Hz (high-performance mode)
0011	52 Hz (low-power mode)	52 Hz (high-performance mode)
0100	104 Hz (low-power mode)	104 Hz (high-performance mode)
0101	208 Hz (low-power mode)	208 Hz (high-performance mode)
0110	416 Hz (high-performance mode)	416 Hz (high-performance mode)
0111	833 Hz (high-performance mode)	833 Hz (high-performance mode)
1000	1667 Hz (high-performance mode)	1667 Hz (high-performance mode)
1001	3333 Hz (high-performance mode)	3333 Hz (high-performance mode)
1010	6667 Hz (high-performance mode)	6667 Hz (high-performance mode)

Table 9. Power consumption (@ Vdd = 1.8 V, T = 25 °C) shows the typical values of power consumption for the different operating modes.

Table 9. Power consumption (@ Vdd = 1.8 V, T = 25 °C)

ODR [Hz]	Accelerometer only	Gyroscope only	Combo [accelerometer + gyroscope]
Power-down mode	-	-	6 µA
Sleep mode	-	285 µA	-
1.6 Hz (low-power mode)	15 µA	-	-
12.5 Hz (low-power mode)	20 µA	290 µA	300 µA
26 Hz (low-power mode)	25 µA	300 µA	310 µA
52 Hz (low-power mode)	37 µA	320 µA	330 µA
104 Hz (low-power mode)	55 µA	350 µA	375 µA
208 Hz (low-power mode)	97 µA	410 µA	465 µA
12.5 Hz (high-performance mode)	180 µA	490 µA	595 µA
26 Hz (high-performance mode)	180 µA	490 µA	595 µA
52 Hz (high-performance mode)	180 µA	490 µA	595 µA
104 Hz (high-performance mode)	180 µA	490 µA	595 µA
208 Hz (high-performance mode)	180 µA	490 µA	595 µA
416 Hz (high-performance mode)	180 µA	490 µA	595 µA
833 Hz (high-performance mode)	180 µA	490 µA	595 µA
1667 Hz (high-performance mode)	180 µA	490 µA	595 µA
3333 Hz (high-performance mode)	180 µA	490 µA	595 µA
6667 Hz (high-performance mode)	180 µA	490 µA	595 µA

3.1 Power-down mode

When the accelerometer/gyroscope is in power-down mode, almost all internal blocks of the device are switched off to minimize power consumption. The digital interfaces (I²C and SPI) are still active to allow communication with the device. The content of the configuration registers is preserved and the output data registers are not updated, keeping the last data sampled in memory before going into power-down mode.

3.2 High-performance mode

In high-performance mode, all accelerometer/gyroscope circuitry is always on and data are generated at the data rate selected through the ODR_XL/ODR_G bits.

Data interrupt generation is active.

3.3 Low-power mode

While high-performance mode guarantees the best performance in terms of noise, low-power mode further reduces the current consumption. The accelerometer/gyroscope data reading chain is automatically turned on and off to save power. In the gyroscope device, only the driving circuitry is always on.

Data interrupt generation is active.

3.4 Gyroscope sleep mode

While the gyroscope is in sleep mode the circuitry that drives the oscillation of the gyroscope mass is kept active. Compared to gyroscope power-down, turn-on time from sleep mode to low-power/high-performance mode is drastically reduced.

If the gyroscope is not configured in power-down mode, it enters sleep mode when the sleep mode (SLEEP_G) bit of the CTRL4_C register is set to 1, regardless of the selected gyroscope ODR. If the gyroscope is configured in power-down mode, the SLEEP_G bit configuration is ignored.

3.5 Connection modes

The ISM330IS offers two different connection modes, described in detail in this document:

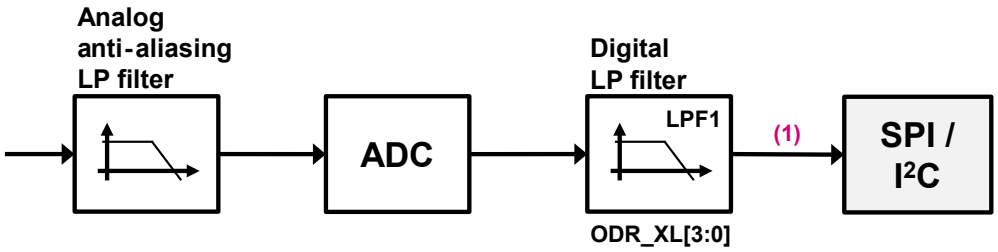
- **Mode 1:** it is the connection mode enabled by default. The I²C slave interface or SPI (3- / 4-wire) serial interface is available.
- **Mode 2:** it is the sensor hub mode. The I²C slave interface or SPI (3- / 4-wire) serial interface and I²C interface master for external sensor connections are available. This connection mode is described in Section 6 Mode 2 - sensor hub mode.

3.6 Accelerometer bandwidth

The accelerometer sampling chain is represented by a cascade of three main blocks: an analog anti-aliasing low-pass filter, an ADC converter, and a digital low-pass filter.

As shown in Figure 2. Accelerometer filtering chain, the analog signal coming from the mechanical parts is filtered by an analog low-pass anti-aliasing filter before being converted by the ADC. The anti-aliasing filter is enabled in high-performance mode only.

Figure 2. Accelerometer filtering chain



(1)
The cutoff value of the LPF1 output is:
- **ODR / 2 in high-performance mode**
- **700 Hz in low-power mode**

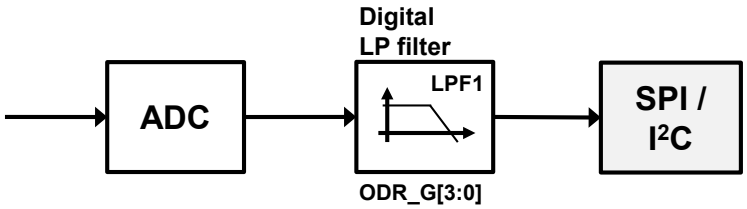
The digital low-pass filter LPF1 cannot be configured by the user and its cutoff frequency depends on the accelerometer mode selected:

- ODR / 2 when the accelerometer is configured in high-performance mode
- 700 Hz when the accelerometer is configured in low-power mode

3.7 Gyroscope bandwidth

The gyroscope filtering chain configuration is shown in Figure 3. Gyroscope filtering chain. The analog signal coming from the mechanical parts is converted by the ADC and the converted signal is then processed by a digital low-pass filter (LPF1).

Figure 3. Gyroscope filtering chain



The digital LPF1 filter cannot be configured by the user and its cutoff frequency depends on the selected gyroscope ODR. When the gyroscope ODR is equal to 6667 Hz, the LPF1 filter is bypassed.

The gyroscope bandwidth is summarized in the following table.

Table 10. Gyroscope bandwidth

Gyroscope ODR [Hz]	Bandwidth [Hz]
12.5	4
26	8
52	17
104	33
208	67
416	137
833	312
1667	988
3333	1161
6667	1250

3.8 Accelerometer and gyroscope turn-on/off time

The accelerometer reading chain contains low-pass filtering to improve signal-to-noise performance and to reduce aliasing effects. For this reason, it is necessary to take into account the settling time of the filters when the accelerometer power mode is switched or when the accelerometer ODR is changed.

The maximum overall turn-on/off time in order to switch accelerometer power modes or accelerometer ODR is shown in Table 11. Accelerometer turn-on/off time

Note: The accelerometer ODR timing is not impacted by power mode changes (a new configuration is effective after the completion of the current period).

Table 11. Accelerometer turn-on/off time

Starting mode	Target mode	Max turn-on/off time(1)
Power-down	Low-power	See Table 12. Accelerometer samples to be discarded
Power-down	High-performance	See Table 12. Accelerometer samples to be discarded
Low-power	High-performance	See Table 12. Accelerometer samples to be discarded + discard 1 additional sample
Low-power	Low-power (ODR change)	See Table 12. Accelerometer samples to be discarded
High-performance	Low-power	See Table 12. Accelerometer samples to be discarded + discard 1 additional sample
High-performance	High-performance @ ODR < 6667 Hz	Discard 3 samples
High-performance	High-performance @ ODR = 6667 Hz	Discard 7 samples
Low-power / high-performance	Power-down	1 µs

1. *Settling time @ 99% of the final value*

Table 12. Accelerometer samples to be discarded

Target mode accelerometer ODR [Hz]	Number of samples to be discarded
1.6 (low-power)	0 (first sample correct)
12.5 (low-power)	0 (first sample correct)
26 (low-power)	0 (first sample correct)
52 (low-power)	0 (first sample correct)
104 (low-power)	0 (first sample correct)
208 (low-power)	0 (first sample correct)
12.5 (high-performance)	1
26 (high-performance)	1
52 (high-performance)	1
104 (high-performance)	1
208 (high-performance)	1
416 (high-performance)	1
833 (high-performance)	1
1667 (high-performance)	4
3333 (high-performance)	10
6667 (high-performance)	32

Turn-on/off time has to be considered also for the gyroscope sensor when switching its modes or when the gyroscope ODR is changed.

The maximum overall turn-on/off time in order to switch gyroscope power modes or gyroscope ODR is shown in Table 13. Gyroscope turn-on/off time.

Note: The gyroscope ODR timing is not impacted by power mode changes (a new configuration is effective after the completion of the current period).

Table 13. Gyroscope turn-on/off time

Starting mode	Target mode	Max turn-on/off time (1)
Power-down	Sleep	70 ms
Power-down	Low-power	70 ms + discard 1 sample
Power-down	High-performance	70 ms + see Table 14. Gyroscope samples to be discarded
Sleep	Low-power	Discard 1 sample
Sleep	High-performance	See Table 14. Gyroscope samples to be discarded
Low-power	High-performance	Discard 2 samples
Low-power	Low-power (ODR change)	Discard 1 sample
High-performance	Low-power	Discard 1 sample
High-performance	High-performance (ODR change)	Discard 2 samples
Low-power / high-performance	Power-down	1 µs if both accelerometer and gyroscope in PD 300 µs if accelerometer not in PD

1. *Settling time @ 99% of the final value*

Table 14. Gyroscope samples to be discarded

Gyroscope ODR [Hz]	Number of samples to be discarded
12.5 Hz	2
26 Hz	3
52 Hz	3
104 Hz	3
208 Hz	3
416 Hz	3
833 Hz	3
1667 Hz	135
3333 Hz	270
6667 Hz	540

3.9 Reboot and software reset

After the device is powered up, it performs a 10 ms (maximum) boot procedure to load the trimming parameters. After the boot is completed, both the accelerometer and the gyroscope are automatically configured in power-down mode. During the boot time the registers are not accessible.

After power-up, the trimming parameters can be reloaded by setting the BOOT bit of the CTRL3_C register to 1.

No toggle of the device power lines is required and the content of the device control registers is not modified. If a reset to the default value of the control registers is required, it can be performed by setting the SW_RESET bit of the CTRL3_C register to 1. When this bit is set to 1, the following registers are reset to their default value:

- FUNC_CFG_ACCESS (01h)
- PIN_CTRL (02h)
- INT1_CTRL (0Dh) and INT2_CTRL (0Eh)
- CTRL1_XL (10h) through CTRL10_C (19h)

The software reset procedure takes a maximum of 50 µs. The status of the reset is signaled by the status of the SW_RESET bit of the CTRL3_C register. Once the reset is completed, this bit is automatically set low.

The status of the boot is signaled by the status of the BOOT bit of the CTRL3_C register. Once the reboot is completed, this bit is automatically set low. The boot status signal can also be driven to the INT1 interrupt pin by setting the INT1_BOOT bit of the INT1_CTRL register to 1. This signal is set high while the boot is running and it is set low again at the end of the boot procedure.

The reboot flow is as follows:

1. Set both the accelerometer and gyroscope in power-down mode
2. Set the INT1_BOOT bit of the INT1_CTRL register to 1 [optional]
3. Set the BOOT bit of the CTRL3_C register to 1
4. Monitor reboot status, three possibilities:
   a. Wait 10 ms
   b. Monitor the INT1 pin until it returns to 0 (step 2. is mandatory in this case)
   c. Poll the BOOT bit of CTRL3_C until it returns to 0

The software reset flow is as follows:

1. Set both the accelerometer and gyroscope in power-down mode
2. Set the SW_RESET bit of CTRL3_C to 1
3. Monitor the software reset status, two possibilities:
   a. Wait 50 µs
   b. Poll the SW_RESET bit of CTRL3_C until it returns to 0

In order to avoid conflicts, the reboot and the software reset must not be executed at the same time (do not set to 1 at the same time both the BOOT bit and SW_RESET bit of CTRL3_C register). The above flows must be performed serially.

For the boot and software reset of the ISPU core, refer to Section 9 ISPU.

4 Mode 1 - reading output data

4.1 Startup sequence

Once the device is powered up, it automatically downloads the calibration coefficients from the embedded flash to the internal registers. When the boot procedure is completed, that is, after approximately 10 ms, the accelerometer and gyroscope automatically enter power-down mode.

To turn on the accelerometer and gather acceleration data, it is necessary to select an output data rate setting different from power-down through the CTRL1_XL register.

The following general-purpose sequence can be used to configure the accelerometer:

1. Write INT1_CTRL = 01h // Accelerometer data-ready interrupt on INT1
2. Write CTRL1_XL = 60h // Accelerometer 416 Hz (high-performance mode)

To turn on the gyroscope and gather angular rate data, it is necessary to select an output data rate setting different from power-down through the CTRL2_G register.

The following general-purpose sequence can be used to configure the gyroscope:

1. Write INT1_CTRL = 02h // Gyroscope data-ready interrupt on INT1
2. Write CTRL2_G = 60h // Gyroscope = 416 Hz (high-performance mode)

4.2 Using the status register

The device is provided with a STATUS_REG register which can be polled to check when a new set of data is available. The XLDA bit is set to 1 when a new set of data is available at accelerometer output. The GDA bit is set to 1 when a new set of data is available at the gyroscope output.

For the accelerometer (the gyroscope is similar), the read of the output registers can be performed as follows:

1. Read STATUS_REG
2. If XLDA = 0, then go to 1
3. Read OUTX_L_A
4. Read OUTX_H_A
5. Read OUTY_L_A
6. Read OUTY_H_A
7. Read OUTZ_L_A
8. Read OUTZ_H_A
9. Data processing
10. Go to 1

4.3 Using the data-ready signal

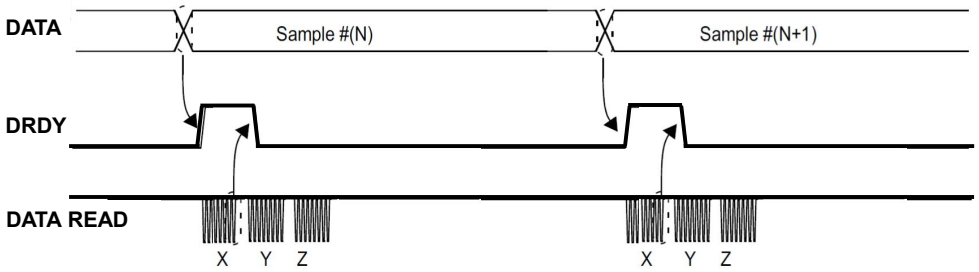
The device can be configured to have a hardware signal to determine when a new set of measurement data is available to be read.

For the accelerometer sensor, the data-ready signal is represented by the XLDA bit of the STATUS_REG register. The signal can be driven to the INT1 pin by setting the INT1_DRDY_XL bit of the INT1_CTRL register to 1 and to the INT2 pin by setting the INT2_DRDY_XL bit of the INT2_CTRL register to 1.

For the gyroscope sensor, the data-ready signal is represented by the GDA bit of the STATUS_REG register. The signal can be driven to the INT1 pin by setting the INT1_DRDY_G bit of the INT1_CTRL register to 1 and to the INT2 pin by setting the INT2_DRDY_G bit of the INT2_CTRL register to 1.

The data-ready signal rises to 1 when a new set of data has been generated and it is available to be read. The data-ready signal can be either latched or pulsed. If the DRDY_PULSED bit of the DRDY_PULSED_REG register is set to 0 (default value), then the data-ready signal is latched and the interrupt is reset when the higher part of one axis is read (registers 29h, 2Bh, 2Dh for the accelerometer; registers 23h, 25h, 27h for the gyroscope). If the DRDY_PULSED bit of the DRDY_PULSED_REG register is set to 1, then the data-ready is pulsed and the duration of the pulse observed on the interrupt pins is 75 µs. Pulsed mode is not applied to the XLDA and GDA bits which are always latched.

Figure 4. Data-ready signal (DRDY_PULSED = 0)



4.4 Using the block data update (BDU) feature

If reading the accelerometer/gyroscope data is not synchronized with either the XLDA/GDA bits in the STATUS_REG register or with the DRDY signal driven to the INT1/INT2 pins, it is strongly recommended to set the BDU (block data update) bit to 1 in the CTRL3_C register.

This feature avoids reading values (most significant and least significant parts of output data) related to different samples. In particular, when the BDU is activated, the data registers related to each axis always contain the most recent output data produced by the device, but, in case the read of a given pair (that is, OUTX_H_A(G) and OUTX_L_A(G), OUTY_H_A(G) and OUTY_L_A(G), OUTZ_H_A(G) and OUTZ_L_A(G)) is initiated, the refresh for that pair is blocked until both the MSB and LSB of the data are read.

Note: BDU only guarantees that the LSB and MSB have been sampled at the same moment. For example, if the reading speed is too slow, X and Y can be read at T1 and Z sampled at T2.

4.5 Understanding output data

The measured acceleration data are sent to the OUTX_H_A, OUTX_L_A, OUTY_H_A, OUTY_L_A, OUTZ_H_A, and OUTZ_L_A registers. These registers contain, respectively, the most significant part and the least significant part of the acceleration signals acting on the X, Y, and Z axes.

The measured angular rate data are sent to the OUTX_H_G, OUTX_L_G, OUTY_H_G, OUTY_L_G, OUTZ_H_G, and OUTZ_L_G registers. These registers contain, respectively, the most significant part and the least significant part of the angular rate signals acting on the X, Y, and Z axes.

The complete output data for the X, Y, Z axes is given by the concatenation OUTX_H_A(G) & OUTX_L_A(G), OUTY_H_A(G) & OUTY_L_A(G), OUTZ_H_A(G) & OUTZ_L_A(G) and it is expressed as a two's complement number.

Both acceleration data and angular rate data are represented as 16-bit numbers. In order to translate them to their corresponding physical representation, a sensitivity parameter must be applied. This sensitivity value depends on the selected full-scale range (refer to the datasheet). In detail:

- Each acceleration sample must be multiplied by the proper sensitivity parameter LA_So (linear acceleration sensitivity expressed in mg/LSB) in order to obtain the corresponding value in mg.
- Each angular rate sample must be multiplied by the proper sensitivity parameter G_So (angular rate sensitivity expressed in mdps/LSB) in order to obtain the corresponding value in mdps.

4.5.1 Examples of output data

Table 15. Output data registers content vs. acceleration (FS_XL = ±2 *g*) provides a few basic examples of the accelerometer data that is read in the data registers when the device is subject to a given acceleration.

Table 16. Output data registers content vs. angular rate (FS_G = ±250 dps) provides a few basic examples of the gyroscope data that is read in the data registers when the device is subject to a given angular rate.

The values listed in the following tables are given under the hypothesis of perfect device calibration (that is, no offset, no gain error, and so forth).

Table 15. Output data registers content vs. acceleration (FS_XL = ±2 *g*)

Acceleration values	Register address	
	OUTX_H_A (29h)	OUTX_L_A (28h)
0 *g*	00h	00h
350 m*g*	16h	69h
1 *g*	40h	09h
-350 m*g*	E9h	97h
-1 *g*	BFh	F7h

Table 16. Output data registers content vs. angular rate (FS_G = ±250 dps)

Angular rate values	Register address	
	OUTX_H_G (23h)	OUTX_L_G (22h)
0 dps	00h	00h
100 dps	2Ch	A4h
200 dps	59h	49h
-100 dps	D3h	5Ch
-200 dps	A6h	B7h

5 Timestamp

Together with sensor data the device can provide timestamp information.

In order to enable this functionality, the TIMESTAMP_EN bit of the CTRL10_C register has to be set to 1. The time step count is given by the concatenation of the TIMESTAMP3 & TIMESTAMP2 & TIMESTAMP1 & TIMESTAMP0 registers and is represented as a 32-bit unsigned number.

The nominal timestamp resolution is 25 µs. It is possible to get the actual timestamp resolution value through the FREQ_FINE[7:0] bits, representing a value as a 8-bit number in two's complement, of the INTERNAL_FREQ_FINE register, which contains the difference in percentage of the actual ODR (and timestamp rate) with respect to the nominal value.

$$t_{actual}[s] = \frac{1}{40000 \cdot (1 + 0.0015 \cdot FREQ\_FINE)}$$

Similarly, it is possible to get the actual output data rate by using the following formula:

$$ODR_{actual}[Hz] = \frac{6667 + 0.0015 \cdot FREQ\_FINE \cdot 6667}{ODR_{coeff}}$$

where the ODR_{coeff} values are indicated in the table below.

Table 17. ODR_{coeff} values

Selected ODR [Hz]	ODR_{coeff}
12.5	512
26	256
52	128
104	64
208	32
416	16
833	8
1667	4
3333	2
6667	1

If both the accelerometer and the gyroscope are in power-down mode, the timestamp counter does not work and the timestamp value is frozen at the last value.

When the maximum value 4294967295 LSB (equal to FFFFFFFFh) is reached corresponding to approximately 30 hours, the counter is automatically reset to 00000000h and continues to count. The timer count can be reset to zero at any time by writing the reset value AAh in the TIMESTAMP2 register.

The TIMESTAMP_ENDCOUNT bit of the STATUS_REG goes high 6.4 ms before the occurrence of a timestamp overrun condition. This flag is reset when the STATUS_REG register is read. It is also possible to route this signal to the INT2 pin (75 µs duration pulse) by setting the INT2_TIMESTAMP bit of MD2_CFG to 1.

6 Mode 2 - sensor hub mode

The hardware flexibility of the ISM330IS allows connecting the pins with different mode connections to external sensors to expand functionalities such as adding a sensor hub. When sensor hub mode (mode 2) is enabled, both the primary I²C/SPI (3- and 4-wire) slave interface and the I²C master interface for the connection of external sensors are available. Mode 2 connection mode is described in detail in the following paragraphs.

6.1 Sensor hub mode description

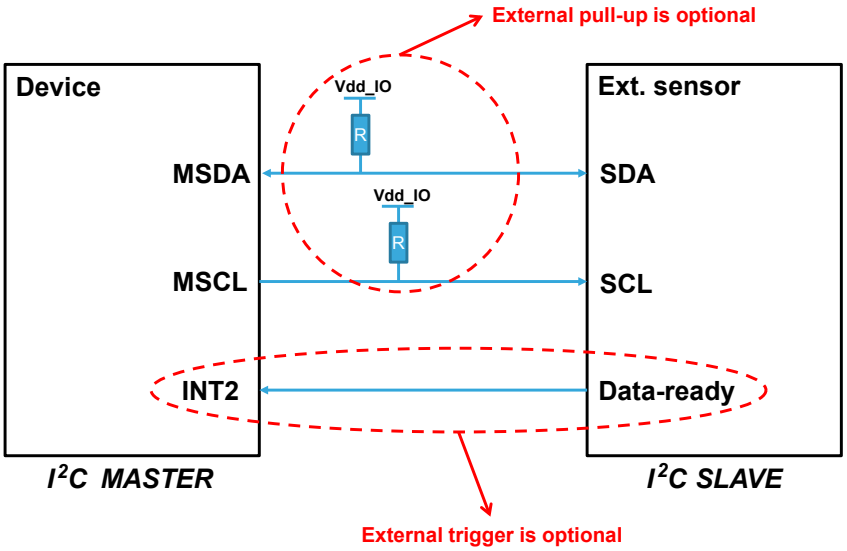
In sensor hub mode (mode 2) up to four external sensors can be connected to the I²C master interface of the device. The sensor hub trigger signal can be synchronized with the accelerometer/gyroscope data-ready signal (up to 104 Hz). In this configuration, the sensor hub ODR can be configured through the SHUB_ODR_[1:0] bits of the SLV0_CONFIG register. Alternatively, an external signal connected to the INT2 pin can be used as the sensor hub trigger. In this second case, the maximum ODR supported for external sensors depends on the number of read / write operations that can be executed between two consecutive trigger signals.

On the sensor hub trigger signal, all the write and read I²C operations configured through the registers SLVx_ADD, SLVx_SUBADD, SLVx_CONFIG and DATAWRITE_SLV0 are performed sequentially from external sensor 0 to external sensor 3 (depending on the external sensors enabled through the AUX_SENS_ON_[1:0] field in the MASTER_CONFIG register).

If both the accelerometer and the gyroscope are in power-down mode, the sensor hub does not work.

All external sensors have to be connected in parallel to the MSDA/MSCL pins of the device, as illustrated in Figure 5. External sensor connections in mode 2 for a single external sensor. External pull-up resistors and the external trigger signal connection are optional and depend on the configuration of the registers.

Figure 5. External sensor connections in mode 2



6.2 Sensor hub mode registers

The sensor hub configuration registers and output registers are accessible when the bit SHUB_REG_ACCESS of the FUNC_CFG_ACCESS register is set to 1. After setting the SHUB_REG_ACCESS bit to 1, only sensor hub registers are available. In order to guarantee the correct register mapping for other operations, after the sensor hub configuration or output data reading, the SHUB_REG_ACCESS bit of the FUNC_CFG_ACCESS register must be set to 0.

The MASTER_CONFIG register has to be used for the configuration of the I²C master interface.

A set of registers SLVx_ADD, SLVx_SUBADD, SLVx_CONFIG is dedicated to the configuration of the 4 slave interfaces associated to the 4 connectable external sensors. An additional register, DATAWRITE_SLV0, is associated to slave #0 only. It has to be used to implement the write operations.

Finally, 18 registers (from SENSOR_HUB_1 to SENSOR_HUB_18) are available to store the data read from the external sensors.

6.2.1 MASTER_CONFIG (14h)

This register is used to configure the I²C master interface.

Table 18. MASTER_CONFIG register

b7	b6	b5	b4	b3	b2	b1	b0
RST_MASTER_REGS	WRITE_ONCE	START_CONFIG	PASS_THROUGH_MODE	SHUB_PU_EN	MASTER_ON	AUX_SENS_ON_1	AUX_SENS_ON_0

- RST_MASTER_REGS bit is used to reset the I²C master interface, configuration and output registers. It must be manually asserted and de-asserted.
- WRITE_ONCE bit is used to limit the write operations on slave 0 to only one occurrence (avoiding to repeat the same write operation multiple times). If this bit is not asserted, a write operation is triggered at each ODR.

Note: The WRITE_ONCE bit must be set to 1 if slave 0 is used for read transactions.

- START_CONFIG bit selects the sensor hub trigger signal.
  - When this bit is set to 0, the accelerometer/gyroscope sensor has to be active (not in power-down mode) and the sensor hub trigger signal is the accelerometer/gyroscope data-ready signal, with a frequency defined by the SHUB_ODR_[1:0] bits of the SLV0_CONFIG register (up to 104 Hz).
  - When this bit is set to 1, at least one sensor between the accelerometer and the gyroscope has to be active and the sensor hub trigger signal is the INT2 pin. In fact, when both the MASTER_ON bit and START_CONFIG bit are set to 1, the INT2 pin is configured as an input signal. In this case, the INT2 pin has to be connected to the data-ready pin of the external sensor (Figure 5. External sensor connections in mode 2) in order to trigger the read/write operations on the external sensor registers. Sensor hub interrupt from INT2 is 'high-level triggered' (not programmable).

Note: In case of external trigger signal usage (START_CONFIG = 1), if the INT2 pin is connected to the data-ready pin of the external sensor (Figure 5. External sensor connections in mode 2) and the latter is in power-down mode, then no data-ready signal can be generated by the external sensor. For this reason, the initial configuration of the external sensor register has to be performed using the internal trigger signal (START_CONFIG = 0). After the external sensor is activated and the data-ready signal is available, the external trigger signal can be used by switching the START_CONFIG bit to 1.

- PASS_THROUGH_MODE bit is used to enable/disable the I²C interface pass-through. When this bit is set to 1, the main I²C line (for example, connected to an external microcontroller) is short-circuited with the auxiliary one, in order to implement a direct access to the external sensor registers. See Section 6.3 Sensor hub pass-through feature for details.
- SHUB_PU_EN bit enables/disables the internal pull-up on the I²C master line. When this bit is set to 0, the internal pull-up is disabled and the external pull-up resistors on the MSDA/MSCL pins are required, as shown in Figure 5. External sensor connections in mode 2. When this bit is set to 1, the internal pull-up is enabled (regardless of the configuration of the MASTER_ON bit) and the external pull-up resistors on the MSDA/MSCL pins are not required.

- MASTER_ON bit has to be set to 1 to enable the auxiliary I²C master of the device (sensor hub mode). In order to change the sensor hub configuration at runtime or when setting the accelerometer and gyroscope sensor in power-down mode, or when applying the software reset procedure, the I²C master must be disabled, followed by a 300 µs delay. The following procedure must be implemented:

1. Turn off I²C master by setting MASTER_ON = 0
2. Wait 300 µs
3. Change the configuration of the sensor hub registers or set the accelerometer/gyroscope in power-down mode or apply the software reset procedure

- AUX_SENS_ON_[1:0] bits have to be set accordingly to the number of slaves to be used. I²C transactions are performed sequentially from slave 0 to slave 3. The possible values are:
  - 00: one slave
  - 01: two slaves
  - 10: three slaves
  - 11: four slaves

6.2.2 STATUS_MASTER (22h)

The STATUS_MASTER register, similarly to the other sensor hub configurations and output registers, can be read only after setting the SHUB_REG_ACCESS bit of the FUNC_CFG_ACCESS register to 1. The STATUS_MASTER register is also mapped to the STATUS_MASTER_MAINPAGE register, which can be directly read without enabling access to the sensor hub registers.

Table 19. STATUS_MASTER / STATUS_MASTER_MAINPAGE register

b7	b6	b5	b4	b3	b2	b1	b0
WR_ONCE _DONE	SLAVE3_ NACK	SLAVE2_ NACK	SLAVE1_ NACK	SLAVE0_ NACK	0	0	SENS_HUB _ENDOP

- WR_ONCE_DONE bit is set to 1 after a write operation performed with the WRITE_ONCE bit configured to 1 in the MASTER_CONFIG register. This bit can be polled in order to check if the single write transaction has been completed.
- SLAVEx_NACK bits are set to 1 if a “not acknowledge” event happens during the communication with the corresponding slave x.
- SENS_HUB_ENDOP bit reports the status of the I²C master. During the idle state of the I²C master, this bit is equal to 1; it goes to 0 during I²C master read/write operations. When a sensor hub routine is completed, this bit automatically goes to 1 and the external sensor data are available to be read from the SENSOR_HUB_x registers (depending on the configuration of the SLVx_ADD, SLVx_SUBADD, SLVx_CONFIG registers). Information about the status of the I²C master can be driven to the INT1 interrupt pin by setting the INT1_SHUB bit of the MD1_CFG register to 1. This signal goes high on a rising edge of the SENS_HUB_ENDOP signal and it is cleared only if the STATUS_MASTER / STATUS_MASTER_MAINPAGE register is read.

6.2.3 SLV0_ADD (15h), SLV0_SUBADD (16h), SLV0_CONFIG (17h)

The sensor hub registers used to configure the I²C slave interface associated to the first external sensor are described hereafter.

Table 20. SLV0_ADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave0_add6	slave0_add5	slave0_add4	slave0_add3	slave0_add2	slave0_add1	slave0_add0	rw_0

- slave0_add[6:0] bits are used to indicate the I²C slave address of the first external sensor.
- rw_0 bit configures the read/write operation to be performed on the first external sensor (0: write operation; 1: read operation). The read/write operation is executed when the next sensor hub trigger event occurs.

Table 21. SLV0_SUBADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave0_reg7	slave0_reg6	slave0_reg5	slave0_reg4	slave0_reg3	slave0_reg2	slave0_reg1	slave0_reg0

- slave0_reg[7:0] bits are used to indicate the address of the register of the first external sensor to be written (if the rw_0 bit of the SLV0_ADD register is set to 0) or the address of the first register to be read (if the rw_0 bit is set to 1).

Table 22. SLV0_CONFIG register

b7	b6	b5	b4	b3	b2	b1	b0
SHUB_ ODR_1	SHUB_ ODR_0	0	0	0	Slave0 _numop2	Slave0 _numop1	Slave0 _numop0

- SHUB_ODR_[1:0] bits are used to configure the sensor hub output data rate when using internal trigger (accelerometer/gyroscope data-ready signals). The sensor hub output data rate can be configured to four possible values, limited by the ODR of the accelerometer and gyroscope sensors:
  - 00: 104 Hz
  - 01: 52 Hz
  - 10: 26 Hz
  - 11: 12.5 Hz

The maximum allowed value for the SHUB_ODR_[1:0] bits corresponds to the maximum ODR between the accelerometer and gyroscope sensors.

- Slave0_numop[2:0] bits define the number of consecutive read operations to be performed on the first external sensor starting from the register address indicated in the SLV0_SUBADD register.

6.2.4 SLV1_ADD (18h), SLV1_SUBADD (19h), SLV1_CONFIG (1Ah)

The sensor hub registers used to configure the I²C slave interface associated to the second external sensor are described hereafter.

Table 23. SLV1_ADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave1_add6	slave1_add5	slave1_add4	slave1_add3	slave1_add2	slave1_add1	slave1_add0	r_1

- slave1_add[6:0] bits are used to indicate the I²C slave address of the second external sensor.
- r_1 bit enables/disables the read operation to be performed on the second external sensor (0: read operation disabled; 1: read operation enabled). The read operation is executed when the next sensor hub trigger event occurs.

Table 24. SLV1_SUBADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave1_reg7	slave1_reg6	slave1_reg5	slave1_reg4	slave1_reg3	slave1_reg2	slave1_reg1	slave1_reg0

- Slave1_reg[7:0] bits are used to indicate the address of the register of the second external sensor to be read when the r_1 bit of SLV1_ADD register is set to 1.

Table 25. SLV1_CONFIG register

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	Slave1 _numop2	Slave1 _numop1	Slave1 _numop0

- Slave1_numop[2:0] bits define the number of consecutive read operations to be performed on the second external sensor starting from the register address indicated in the SLV1_SUBADD register.

6.2.5 SLV2_ADD (1Bh), SLV2_SUBADD (1Ch), SLV2_CONFIG (1Dh)

The sensor hub registers used to configure the I²C slave interface associated to the third external sensor are described hereafter.

Table 26. SLV2_ADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave2_add6	slave2_add5	slave2_add4	slave2_add3	slave2_add2	slave2_add1	slave2_add0	r_2

- Slave2_add[6:0] bits are used to indicate the I²C slave address of the third external sensor.
- r_2 bit enables/disables the read operation to be performed on the third external sensor (0: read operation disabled; 1: read operation enabled). The read operation is executed when the next sensor hub trigger event occurs.

Table 27. SLV2_SUBADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave2_reg7	slave2_reg6	slave2_reg5	slave2_reg4	slave2_reg3	slave2_reg2	slave2_reg1	slave2_reg0

- Slave2_reg[7:0] bits are used to indicate the address of the register of the third external sensor to be read when the r_2 bit of the SLV2_ADD register is set to 1.

Table 28. SLV2_CONFIG register

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	Slave2 _numop2	Slave2 _numop1	Slave2 _numop0

- Slave2_numop[2:0] bits define the number of consecutive read operations to be performed on the third external sensor starting from the register address indicated in the SLV2_SUBADD register.

6.2.6 SLV3_ADD (1Eh), SLV3_SUBADD (1Fh), SLV3_CONFIG (20h)

The sensor hub registers used to configure the I²C slave interface associated to the fourth external sensor are described hereafter.

Table 29. SLV3_ADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave3_add6	slave3_add5	slave3_add4	slave3_add3	slave3_add2	slave3_add1	slave3_add0	r_3

- Slave3_add[6:0] bits are used to indicate the I²C slave address of the fourth external sensor.
- r_3 bit enables/disables the read operation to be performed on the fourth external sensor (0: read operation disabled; 1: read operation enabled). The read operation is executed when the next sensor hub trigger event occurs.

Table 30. SLV3_SUBADD register

b7	b6	b5	b4	b3	b2	b1	b0
slave3_reg7	slave3_reg6	slave3_reg5	slave3_reg4	slave3_reg3	slave3_reg2	slave3_reg1	slave3_reg0

- Slave3_reg[7:0] bits are used to indicate the address of the register of the fourth external sensor to be read when the r_3 bit of the SLV3_ADD register is set to 1.

Table 31. SLV3_CONFIG register

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	Slave3 _numop2	Slave3 _numop1	Slave3 _numop0

- Slave3_numop[2:0] bits define the number of consecutive read operations to be performed on the fourth external sensor starting from the register address indicated in the SLV3_SUBADD register.

6.2.7 DATAWRITE_SLV0 (21h)

Table 32. DATAWRITE_SLV0 register

b7	b6	b5	b4	b3	b2	b1	b0
Slave0_ dataw7	Slave0_ dataw6	Slave0_ dataw5	Slave0_ dataw4	Slave0_ dataw3	Slave0_ dataw2	Slave0_ dataw1	Slave0_ dataw0

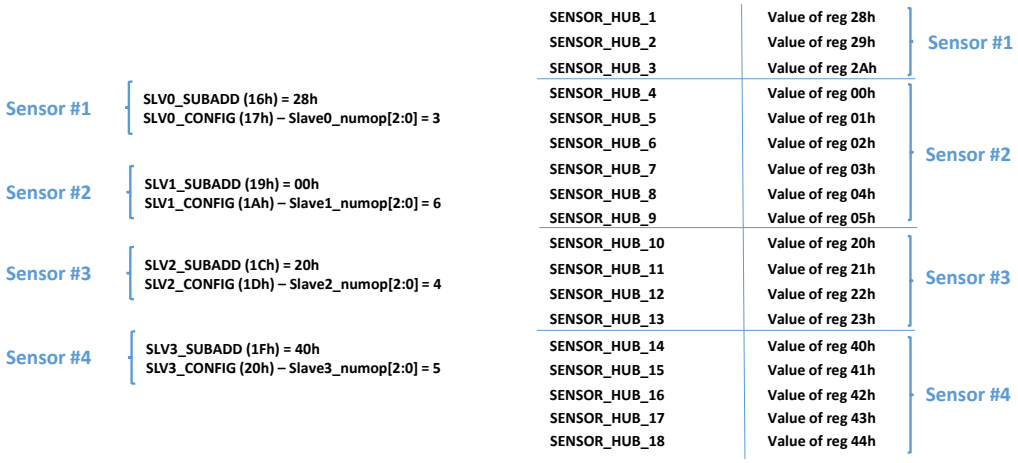
- Slave0_dataw[7:0] bits are dedicated, when the rw_0 bit of SLV0_ADD register is set to 0 (write operation), to indicate the data to be written to the first external sensor at the address specified in the SLV0_SUBADD register.

6.2.8 SENSOR_HUB_x registers

Once the auxiliary I²C master is enabled, for each of the external sensors, it reads a number of registers equal to the value of the Slavex_numop (x = 0, 1, 2, 3) field, starting from the register address specified in the SLVx_SUBADD (x = 0, 1, 2, 3) register. The number of external sensors to be managed is specified in the AUX_SENS_ON_[1:0] bits of the MASTER_CONFIG register.

Read data are consecutively stored (in the same order they are read) in the device registers starting from the SENSOR_HUB_1 register, as in the example in Figure 6. SENSOR_HUB_x allocation example; 18 registers, from SENSOR_HUB_1 to SENSOR_HUB_18, are available to store the data read from the external sensors.

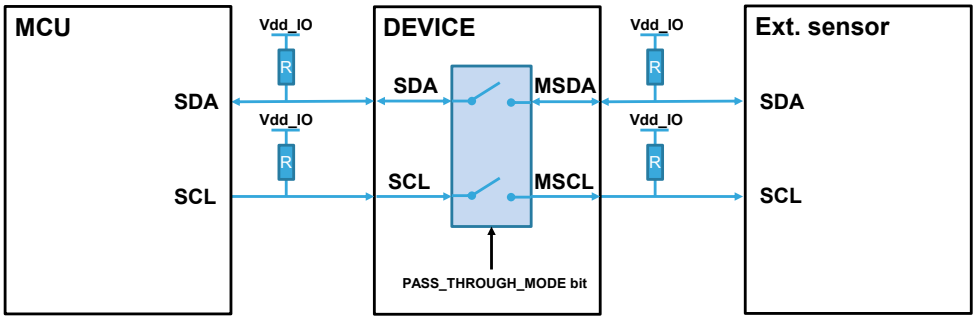
Figure 6. SENSOR_HUB_x allocation example



6.3 Sensor hub pass-through feature

The PASS_THROUGH_MODE bit of the MASTER_CONFIG register is used to enable/disable the I²C interface pass-through. When it is set to 1, the main I²C line (for example, connected to an external microcontroller) is short-circuited with the auxiliary one in order to implement a direct access to the external sensor registers. The pass-through feature for an external device configuration can be used only if the I²C protocol is used on the primary interface. This feature can be used to configure the external sensors.

Figure 7. Pass-through feature



The following procedure can be implemented to enable pass-through mode:

1. If the I²C master is enabled (MASTER_ON = 1), turn it off (set the MASTER_ON bit to 0) and wait 300 µs.
2. If the pull-up on the I²C master line is enabled, disable it (set the SHUB_PU_EN bit of the MASTER_CONFIG register to 0).
3. Enable pass-through mode by setting the PASS_THROUGH_MODE bit to 1.

6.4 Sensor hub mode example

The configuration of the external sensors can be performed using the pass-through feature. This feature can be enabled by setting the PASS_THROUGH_MODE bit of the MASTER_CONFIG register to 1 and implements a direct access to the external sensor registers, allowing quick configuration.

The code provided below gives basic routines to configure a device in sensor hub mode. Three different snippets of code are provided here, in order to present how to easily perform a one-shot write or read operation, using slave 0, and how to set up slave 0 for continuously reading external sensor data.

The PASS_THROUGH_MODE bit is disabled in all these routines, in order to be as generic as possible.

The **one-shot read routine** (using internal trigger) is described below. For simplicity, the routine uses the accelerometer configured at 104 Hz, without external pull-ups on the I²C auxiliary bus.

1.	Write 40h to FUNC_CFG_ACCESS	// Enable access to sensor hub registers
2.	Write EXT_SENS_ADDR | 01h to SLV0_ADD	// Configure external device address (EXT_SENS_ADDR) // Enable read operation (rw_0 = 1)
3.	Write REG to SLV0_SUBADD	// Configure address (REG) of the register to be read
4.	Write 01h to SLV0_CONFIG	// Read one byte, SHUB_ODR = 104 Hz

5.	Write 4Ch to MASTER_CONFIG	// WRITE_ONCE is mandatory for read
		// I²C master enabled, using slave 0 only
		// I²C pull-ups enabled on MSDA and MSCL

6.	Write 00h to FUNC_CFG_ACCESS	// Disable access to sensor hub registers
7.	Read OUTX_H_A register	// Clear accelerometer data-ready XLDA
8.	Poll STATUS_REG, until XLDA = 1	// Wait for sensor hub trigger
9.	Poll STATUS_MASTER_MAINPAGE, until SENS_HUB_ENDOP = 1	// Wait for sensor hub read transaction
10.	Write 40h to FUNC_CFG_ACCESS	// Enable access to sensor hub registers
11.	Write 08h to MASTER_CONFIG	// I²C master disable
12.	Wait 300 µs	
13.	Read SENSOR_HUB_1 register	// Retrieve the output of the read operation
14.	Write 00h to FUNC_CFG_ACCESS	// Disable access to sensor hub registers

The one-shot routine can be easily changed to setup the device for **continuous reading** of external sensor data:

1.	Write 40h to FUNC_CFG_ACCESS	// Enable access to sensor hub registers
2.	Write EXT_SENS_ADDR | 01h to SLV0_ADD	// Configure external device address (EXT_SENS_ADDR) // Enable read operation (rw_0 = 1)
3.	Write REG to SLV0_SUBADD	// Configure address (REG) of the register to be read
4.	Write 0xh to SLV0_CONFIG	// Read x bytes (up to six), SHUB_ODR = 104 Hz

5.	Write 4Ch to MASTER_CONFIG	// WRITE_ONCE is mandatory for read
		// I²C master enabled, using slave 0 only
		// I²C pull-ups enabled on MSDA and MSCL

6.	Write 00h to FUNC_CFG_ACCESS	// Disable access to sensor hub registers

After the execution of step 6, external sensor data are available to be read in sensor hub output registers.

The **one-shot write routine** (using internal trigger) is described below. For simplicity, the routine uses the accelerometer configured at 104 Hz, without external pull-ups on the I²C auxiliary bus.

1.	Write 40h to FUNC_CFG_ACCESS	// Enable access to sensor hub registers
2.	Write EXT_SENS_ADDR to SLV0_ADD	// Configure external device address (EXT_SENS_ADDR)
		// Enable write operation (rw_0 = 0)
3.	Write REG to SLV0_SUBADD	// Configure address (REG) of the register to be written
4.	Write 00h to SLV0_CONFIG	// SHUB_ODR = 104 Hz
5.	Write VAL to DATAWRITE_SLV0	// Configure value (VAL) to be written in REG
6.	Write 4Ch to MASTER_CONFIG	// WRITE_ONCE enabled for single write
		// I²C master enabled, using slave 0 only
		// I²C pull-ups enabled on MSDA and MSCL
7.	Poll STATUS_MASTER,	// Wait for sensor hub write transaction
	until WR_ONCE_DONE = 1	
8.	Write 08h to MASTER_CONFIG	// I²C master disabled
9.	Wait 300 µs	
10.	Write 00h to FUNC_CFG_ACCESS	// Disable access to sensor hub registers

The following sequence configures the LIS2MDL external magnetometer sensor (refer to its datasheet for additional details) in continuous-conversion mode at 100 Hz (enabling temperature compensation, BDU and offset cancellation features) and reads the magnetometer output registers, saving their values in the SENSOR_HUB_1 to SENSOR_HUB_6 registers.

1.	Write 40h to CTRL1_XL	// Turn on the accelerometer (for trigger signal) at 104 Hz
2.	Perform **one-shot read** with	// Check LIS2MDL WHO_AM_I register
	SLV0_ADD = 3Dh	// LIS2MDL slave address is 3Ch and rw_0=1
	SLV0_SUBADD = 4Fh	// WHO_AM_I register address is 4Fh
3.	Perform **one-shot write** with	// Write LIS2MDL register CFG_REG_A (60h) = 8Ch
	SLV0_ADD = 3Ch	// LIS2MDL slave address is 3Ch and rw_0=0
	SLV0_SUBADD = 60h	// Enable temperature compensation
	DATAWRITE_SLV0 = 8Ch	// Enable magnetometer at 100 Hz ODR in continuous mode
4.	Perform **one-shot write** with	// Write LIS2MDL register CFG_REG_B (61h) = 02h
	SLV0_ADD = 3Ch	// LIS2MDL slave address is 3Ch and rw_0=0
	SLV0_SUBADD = 61h	// Enable magnetometer offset-cancellation
	DATAWRITE_SLV0 = 02h	
5.	Perform **one-shot write** with	// Write LIS2MDL register CFG_REG_B (62h) = 10h
	SLV0_ADD = 3Ch	// LIS2MDL slave address is 3Ch and rw_0=0
	SLV0_SUBADD = 62h	// Enable magnetometer BDU
	DATAWRITE_SLV0 = 10h	
6.	Set up **continuous read** with	// LIS2MDL slave address is 3Ch and rw_0=1
	SLV0_ADD = 3Dh	// Magnetometer output registers start from 68h
	SLV0_SUBADD = 68h	// Set up a continuous 6-byte read from I²C master interface
	SLV0_CONFIG = 06h	

7 Temperature sensor

The ISM330IS is provided with an internal temperature sensor that is suitable for ambient temperature measurement.

If both the accelerometer and the gyroscope sensors are in power-down mode, the temperature sensor is off.

The maximum output data rate of the temperature sensor is 52 Hz and its value depends on how the accelerometer and gyroscope sensors are configured:

- If the gyroscope is in power-down mode:
  - If the accelerometer is configured in low-power mode and its ODR is lower than 52 Hz, the temperature data rate is equal to the configured accelerometer ODR.
  - The temperature data rate is equal to 52 Hz for all other accelerometer configurations.
- If the gyroscope is not in power-down mode, the temperature data rate is equal to 52 Hz, regardless of the accelerometer and gyroscope configuration.

For the temperature sensor, the data-ready signal is represented by the TDA bit of the STATUS_REG register. The signal can be driven to the INT2 pin by setting the INT2_DRDY_TEMP bit of the INT2_CTRL register to 1.

The temperature data is given by the concatenation of the OUT_TEMP_H and OUT_TEMP_L registers and it is represented as a number of 16 bits in two's complement format with a sensitivity of 256 LSB/°C. The output zero level corresponds to 25 °C.

7.1 Example of temperature data calculation

Table 33. Output data registers content vs. temperature provides a few basic examples of the data that is read from the temperature data registers at different ambient temperature values. The values listed in this table are given under the hypothesis of perfect device calibration (that is, no offset, no gain error, and so forth).

Table 33. Output data registers content vs. temperature

Temperature values	Register address	
	OUT_TEMP_H (21h)	OUT_TEMP_L (20h)
0°C	E7h	00h
25°C	00h	00h
50°C	19h	00h

8 Self-test

The embedded self-test functions allow checking the device functionality without moving it.

8.1 Accelerometer self-test

When the accelerometer self-test is enabled, an actuation force is applied to the sensor, simulating a definite input acceleration. In this case, the sensor outputs exhibit a change in their DC levels which are related to the selected full scale through the sensitivity value.

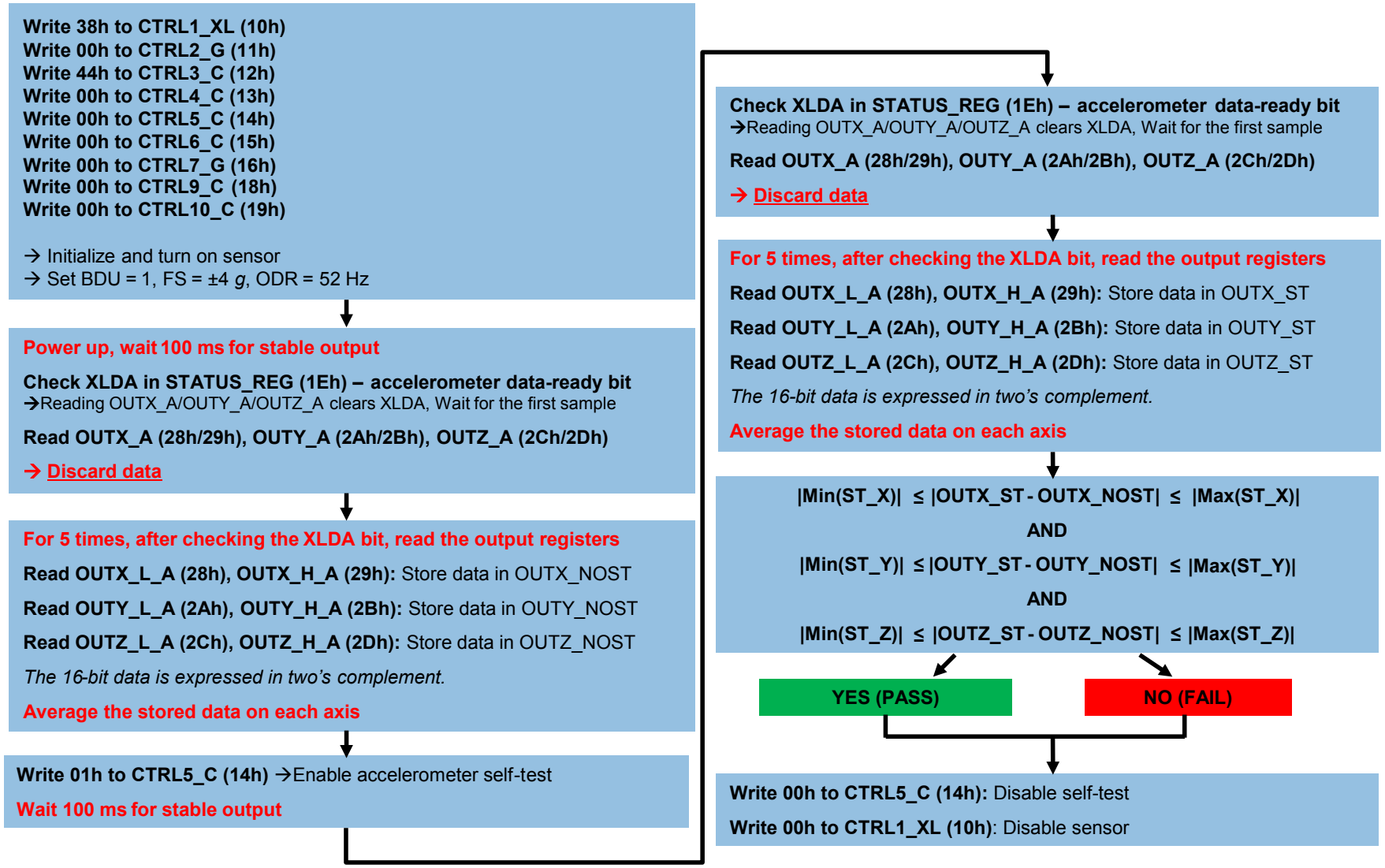
The accelerometer self-test function is off when the ST[1:0] _XL bits of the CTRL5_C register are programmed to 00. It is enabled when the ST[1:0]_XL bits are set to 01 (positive sign self-test) or 10 (negative sign self-test).

When the accelerometer self-test is activated, the sensor output level is given by the algebraic sum of the signals produced by the acceleration acting on the sensor and by the electrostatic test-force.

The complete accelerometer self-test procedure is indicated in Figure 8. Accelerometer self-test procedure.

Figure 8. **Accelerometer self-test procedure**

Note: Keep the device still during the self-test procedure.



8.2 Gyroscope self-test

The gyroscope self-test allows testing the mechanical and electrical parts of the gyroscope sensor: when it is activated, an actuation force is applied to the sensor, emulating a definite Coriolis force and the seismic mass is moved by means of this electrostatic test-force. In this case, the sensor output exhibits an output change.

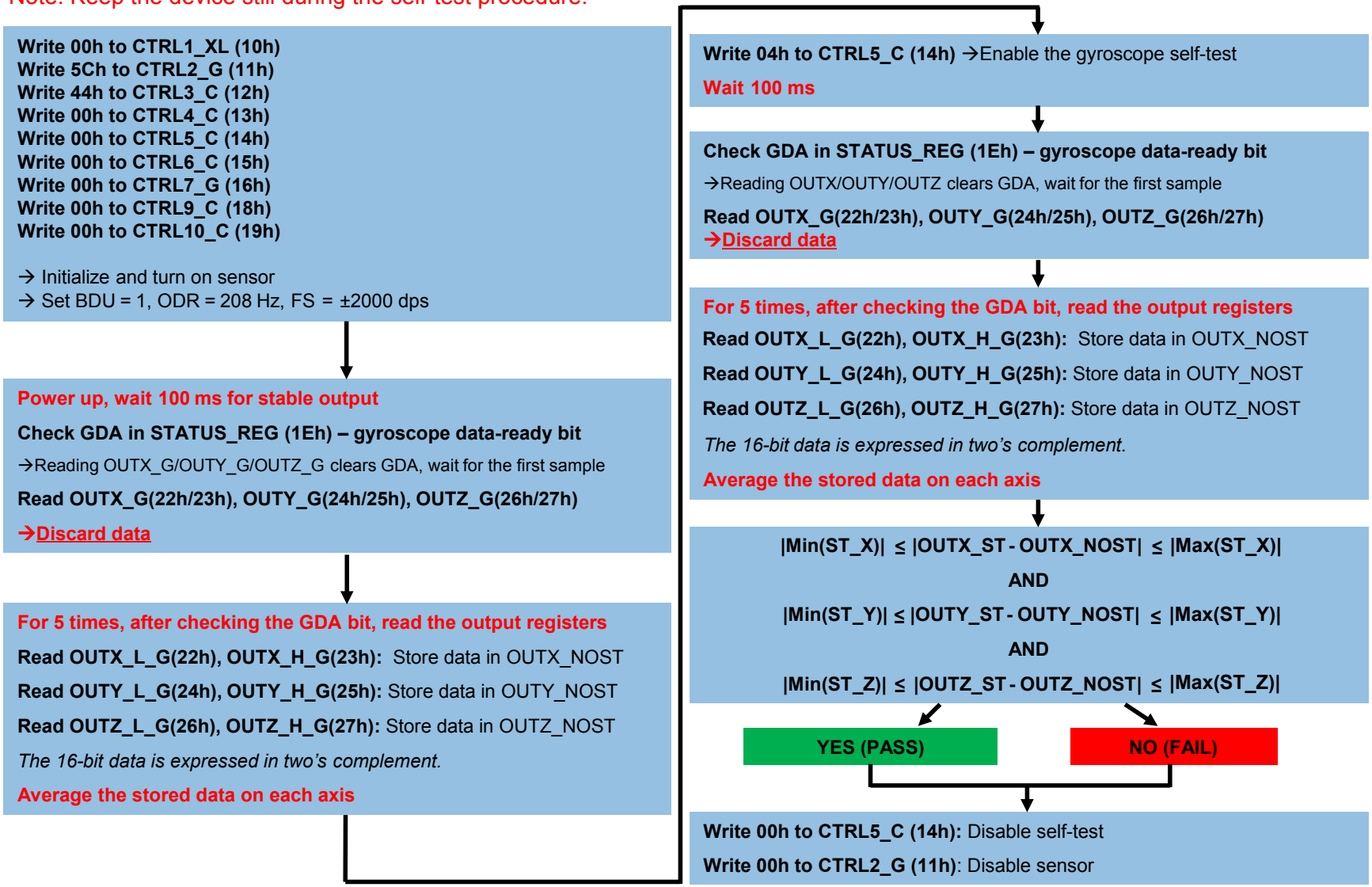
The gyroscope self-test function is off when the ST[1:0]_G bits of the CTRL5_C register are programmed to 00. It is enabled when the ST[1:0]_G bits are set to 01 (positive sign self-test) or 11 (negative sign self-test).

When the gyroscope self-test is active, the sensor output level is given by the algebraic sum of the signals produced by the angular rate acting on the sensor and by the electrostatic test-force.

The complete gyroscope self-test procedure is indicated in Figure 9. Gyroscope self-test procedure.

Figure 9. **Gyroscope self-test procedure**

Note: Keep the device still during the self-test procedure.



9 ISPU

The ISPU (intelligent sensor processing unit) is an embedded programmable core that allows reading sensor data and processing it inside the ISM330IS device and can directly provide, when necessary, the results of said processing to an external microcontroller. The ISPU can run any type of processing, from basic signal processing to artificial intelligence algorithms.

The ISPU in the ISM330IS device is based on the STRED architecture, a proprietary architecture developed by STMicroelectronics, targeting extremely low power consumption and a small silicon area.

The ISPU is based on a 32-bit RISC Harvard architecture and features a minimal floating-point unit (FPU) to accelerate single precision floating-point operations (multiplication, addition, and subtraction). The complete ISPU instruction set is available in Table 38 of Appendix A.

The ISPU is supported by two separate RAMs:

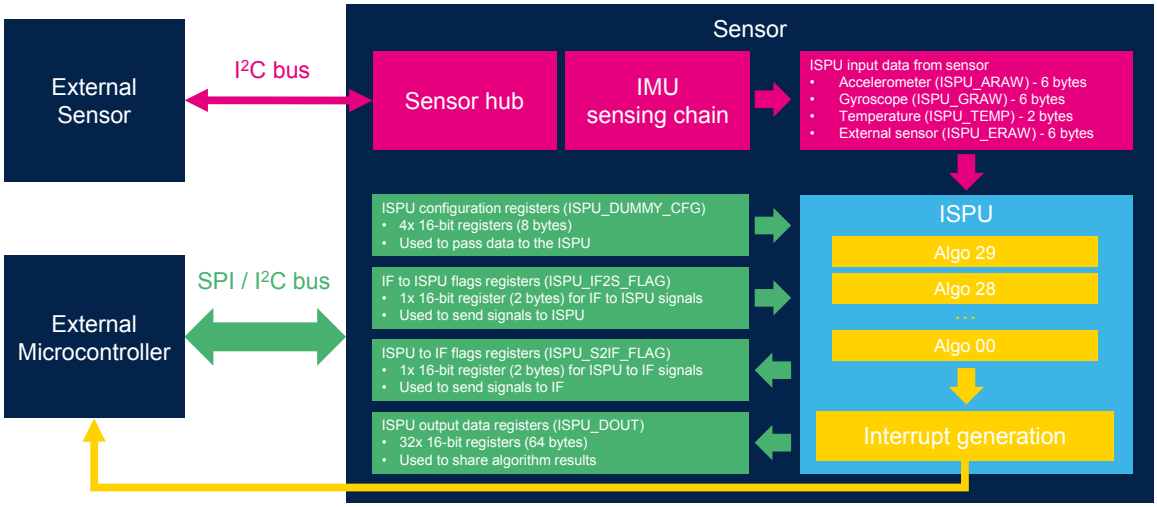
1. 32 KB for code and read-only data
2. 8 KB for variable data

Note that the device does not have any on-board nonvolatile memory to store the program, which means that the program must be reloaded to the device RAM each time power to the device is removed and reapplied.

Note also that the 32 KB RAM is writable from the device interface (I²C or SPI) to load the program, but it is not writable by the ISPU and, as a consequence, cannot be used for storing variable data. The 8 KB RAM instead is writable both from the device interface and by the ISPU.

The ISPU is able to interact both with the sensor via a set of internal registers and with an external microcontroller by communicating through a set of registers accessible both by the ISPU and the external microcontroller over the device interface.

Figure 10. Sensor with ISPU core



The ISPU is designed to be normally in a sleep state, where the clock is disabled. When new data is generated by the sensor, the ISPU is woken up and the new data sample can be processed. As shown in Figure 10, the ISPU can access all the sensor data generated by the device itself (accelerometer, gyroscope, temperature) and the data read from an external sensor through the I²C master interface (see Section 6 Mode 2 - sensor hub mode). The ISPU can also access data written by an external microcontroller in dedicated registers. This data may represent configuration values needed by the processing logic or flags set to request an action to the ISPU. Using all this input data, the ISPU can run up to 30 algorithms and, once the data sample has been processed, it can write the results in the general-purpose output registers, that the external microcontroller can read to retrieve the needed information. The ISPU can also generate interrupt signals on the INT1 and INT2 pins of the device in order to signal to the external microcontroller that there is new relevant data to be read from the output registers or even directly signal the detection of a specific event without the need to read any output data. By using the interrupts, it is possible to keep the external microcontroller in a sleep state for most of the time in order to drastically reduce the power consumption of the system. Once the ISPU has finished all processing for the current data sample and has updated the relevant output registers, then it can be put again in the sleep state to avoid consuming power while waiting for the next data sample.

The ISPU program can be written in standard C code, which allows for high flexibility and reuse of code written for other architectures. It is nonetheless recommended to implement appropriate optimizations for the ISPU architecture if the performance of the processing logic is insufficient to keep up with the sensor data rate or to further reduce power consumption. For example, operations that are not hardware accelerated, like floating-point division, should be avoided as much as possible.

A toolchain is provided on the STMicroelectronics website (ISPU-Toolchain), containing tools for the compilation of the code (based on GNU GCC) and the conversion of binary files into a format suitable to be loaded into the device. Additionally, templates and examples that can be used to quickly start programming the ISPU are available in the X-CUBE-ISPU STM32Cube expansion package.

Note: Support for some of the functions described in this document (for example, processing cycle, algorithm management, interrupt handling) is already available in the ISPU code of the templates and examples.

This document does not describe how to use the software tools mentioned above, but describes every functionality related to the ISPU at the device level.

9.1 Processing cycle and rate configuration

As already introduced, the ISPU is designed for sample-by-sample processing in a cycle:

1. The ISPU is woken up by an internal signal generated by the sensor when new data (from the accelerometer or the gyroscope, not from the external sensor) is available.
2. The code to process the sample is executed.
3. The ISPU is put back to sleep until the next sample is available.

Note: Processing must be completed before the next sample is ready and the signal that triggers the ISPU to wake up is generated. If this condition is not met, it results in one or more missed samples that are not processed by the implemented logic. Refer to Section 9.9 Interrupts for a method to monitor the execution time of the ISPU processing.

The rate at which the ISPU is triggered is configurable and depends on three settings:

- Accelerometer output data rate, configurable by setting the ODR_XL[3:0] field in the CTRL1_XL register. Refer to Table 7 for the available rates.
- Gyroscope output data rate, configurable by setting the ODR_G[3:0] field in the CTRL2_G register. Refer to Table 8 for the available rates.
- ISPU IRQ rate, configurable by setting the ISPU_RATE_[3:0] field in the CTRL9_C register. Refer to Table 34 for the available rates.

Normally, the ISPU IRQ rate represents the rate at which the ISPU is triggered. However, if the accelerometer and gyroscope output data rates are both lower than the ISPU IRQ rate, the rate at which the ISPU is triggered is limited to the faster rate between the accelerometer and gyroscope output data rates. This is due to the fact that the ISPU is triggered by new data samples available from the sensors. The actual ISPU rate can be obtained using the following equation:

$$ISPU\_ACTUAL\_RATE = \min(\max(ODR\_XL, ODR\_G), ISPU\_RATE)$$

Note that, if the ISPU IRQ rate is lower than the sensor output data rate, the sensor data read by the ISPU is downsampled (there is no additional filtering applied to the data, so the signal processed by the ISPU can be potentially affected by aliasing).

Table 34. ISPU IRQ rate selection

ISPU_RATE_[3:0]	Rate
0000	0 Hz (always sleeping)
0001	12.5 Hz
0010	26 Hz
0011	52 Hz
0100	104 Hz
0101	208 Hz
0110	416 Hz
0111	833 Hz
1000	1667 Hz
1001	3333 Hz
1010	6667 Hz
1011-1111	Reserved

The ISPU can be put in the sleep state by attempting to access (via a load or store instruction) an address outside of the address space (see Section 9.2 Memory mapping). This causes the clock of the ISPU to be stopped. This can be achieved, for example, by executing a load from address 0x20000 using the following assembly code:

```
ldb %r0, [0x20000]
```

When the new data sample arrives from the sensor generating the trigger to wake up the ISPU, the clock is restarted and the execution resumes from the next instruction.

9.2 Memory mapping

Memory addresses are expressed as 17-bit values in the ISPU. The address space is subdivided as follows:

- 0x00000 - 0x01FFF → Variable data RAM (8 KB)
- 0x06800 - 0x068FF → Most of the registers, including the input data registers and output registers
- 0x06900 - 0x069FF → Additional registers, including the configuration registers
- 0x10000 - 0x17FFF → Code and read-only data RAM (32 KB)

For a detailed list of all register addresses, refer to Section 2 Registers.

9.3 ISPU interaction registers

Most of the registers used to interact with the ISPU fall under the group named "ISPU interaction registers" (refer to Section 2.1 ISPU interaction registers for the complete list).

In order to access these registers from the device interface, it is necessary to set the ISPU_REG_ACCESS bit in the FUNC_CFG_ACCESS register to 1. Once it is no longer needed to access these registers, the ISPU_REG_ACCESS bit should be set back to 0 in order to restore access to the default registers.

9.4 Sensor data

Both the data generated by the sensor itself and the data read from the external sensor through the sensor hub can be read from the corresponding internal registers only accessible from the ISPU:

- Accelerometer
  - X-axis: ISPU_ARAW_X_L (6880h) contains the least significant byte, ISPU_ARAW_X_H (6881h) contains the most significant byte.
  - Y-axis: ISPU_ARAW_Y_L (6884h) contains the least significant byte, ISPU_ARAW_Y_H (6885h) contains the most significant byte.
  - Z-axis: ISPU_ARAW_Z_L (6888h) contains the least significant byte, ISPU_ARAW_Z_H (6889h) contains the most significant byte.
- Gyroscope
  - X-axis: ISPU_GRAW_X_L (688Ch) contains the least significant byte, ISPU_GRAW_X_H (688Dh) contains the most significant byte.
  - Y-axis: ISPU_GRAW_Y_L (6890h) contains the least significant byte, ISPU_GRAW_Y_H (6891h) contains the most significant byte.
  - Z-axis: ISPU_GRAW_Z_L (6894h) contains the least significant byte, ISPU_GRAW_Z_H (6895h) contains the most significant byte.
- External sensor
  - First byte: ISPU_ERAW_0_L (6898h)
  - Second byte: ISPU_ERAW_0_H (6899h)
  - Third byte: ISPU_ERAW_1_L (689Ch)
  - Fourth byte: ISPU_ERAW_1_H (689Dh)
  - Fifth byte: ISPU_ERAW_2_L (68A0h)
  - Sixth byte: ISPU_ERAW_2_H (68A1h)
- Temperature: ISPU_TEMP_L (68A4h) contains the least significant byte, ISPU_TEMP_H (68A5h) contains the most significant byte.

The values of each axis of the accelerometer and gyroscope, and the temperature value are expressed as 16-bit words in two's complement. Within the ISPU code, they should be interpreted using the *int16_t* type available in the *stdint.h* library. Alternatively, it is possible to read, for each value, two more bytes (available in the registers immediately after the first two bytes) and interpret the four bytes as a 32-bit word in two's complement (*int32_t* type), as a sign extension to 32 bits is applied when the values are written to the registers. The values represent the raw data generated by the sensor, so, if necessary, the sensitivity must be applied to the values by following the same procedure used for the regular sensor output registers.

The external sensor data should be interpreted according to the format produced by the sensor connected to the sensor hub. Note that the sign extension is applied also to the external sensor data for each couple of bytes (for example ISPU_ERAW_0_L and ISPU_ERAW_0_H are treated as one value and extended to 32 bits), but it might be wrong to use depending on the original format of the data. If that is the case, the sign extension should be ignored and only the first 16 bits should be used.

9.5 Configuration

Besides reading the sensor data, it might be useful to receive inputs from the external microcontroller, for example to communicate some configuration parameters values at runtime instead of having to set them at compile time. For this reason, four 16-bit general-purpose registers are available:

- ISPU_DUMMY_CFG_1 (73h-74h / 6974h-6975h)
- ISPU_DUMMY_CFG_2 (75h-76h / 6976h-6977h)
- ISPU_DUMMY_CFG_3 (77h-78h / 6978h-6979h)
- ISPU_DUMMY_CFG_4 (79h-7Ah / 697Ah-697Bh)

These registers are readable and writable from the device interface (that is, from the external microcontroller) and are readable from the ISPU. The registers are general-purpose and any value can be encoded in the 8 bytes available. However, when reading the values from the ISPU, note that:

- The first byte of each 16-bit register (for example, ISPU_DUMMY_CFG_1_L at 6974h) can be read individually, but the second byte (for example, ISPU_DUMMY_CFG_1_H at 6975h) cannot. In order to properly read the second byte, the whole 16-bit register must be read as a single value. The second byte can then be extracted, for example with a shift operation:

```
uint8_t second_byte = *((volatile uint16_t *)ISPU_DUMMY_CFG_1) >> 8;
```

- Bytes that are not contained in the same 16-bit register cannot be read together in one step, but each 16-bit register must be read separately. The final value can then be obtained by reassembling the bytes as needed. For example, to read a 32-bit value contained in ISPU_DUMMY_CFG_1 and ISPU_DUMMY_CFG_2:

```
uint32_t four_bytes_value = ((uint32_t)*((volatile uint16_t *)ISPU_DUMMY_CFG_2) << 16)
| *((volatile uint16_t *)ISPU_DUMMY_CFG_1);
```

If the ISPU_DUMMY_CFG registers are used to set the initial configuration parameters for an algorithm, it is recommended to write these values before enabling the algorithm (explained in Section 9.8 Algorithms), so that they are available for the algorithm initialization code.

An additional register, ISPU_IF2S_FLAG (0Ch-0Dh / 680Ch-680Dh), provides bits that can be set from the device interface (by the external microcontroller) and cleared by the ISPU. This register, contrary to the ISPU_DUMMY_CFG registers, is part of the ISPU interaction registers.

The intended use of the ISPU_IF2S_FLAG register is to raise flags to the ISPU, which can check the register, detect the bits that are set to 1, perform specific actions accordingly, and then clear the bits to signal that the request received from the device interface was served.

In order to set a bit of the ISPU_IF2S_FLAG register, it is sufficient to write 1 to it from the device interface. In order to clear the bit, the ISPU must write again 1 to it. While this might be counterintuitive, writing 1 means requesting the bit to be cleared.

The following code provides an example of how to use the ISPU_IF2S_FLAG within the ISPU:

```
uint8_t flag = *((volatile uint8_t *)ISPU_IF2S_FLAG) & 0x01; // get bit 0 value
if (flag) {
    // serve request from external microcontroller
    *((volatile uint8_t *)ISPU_IF2S_FLAG) = 0x01; // clear bit 0
}
```

9.6 Additional inputs

In addition to the registers described in Section 9.4 Sensor data and Section 9.5 Configuration, there are two more inputs accessible from the ISPU:

- TIMESTAMP0 (40h / 6940h), TIMESTAMP1 (41h / 6941h), TIMESTAMP2 (42h / 6942h) and TIMESTAMP3 (43h / 6943h) provide the timestamp information (see Section 5 Timestamp).
- ISPU_DTIME_0_L (6948h), ISPU_DTIME_0_H (6949h), ISPU_DTIME_1_L (694Ah), and ISPU_DTIME_1_H (694Bh) provide an accurate value of the actual delta time of the device.

The timestamp is expressed as a 32-bit word with a bit resolution of 25 µs (typical).

The delta time is expressed as a 32-bit word encoded as a single-precision floating-point number and represents, in seconds, the actual delta time (time between two consecutive samples) of the sensor with the data rate configured at 104 Hz. The delta time value always refers to the sensor configured at 104 Hz, regardless of the data rate currently set, so the value must be properly rescaled if the delta time for a different data rate is needed (for example, it must be doubled to obtain the delta time for 52 Hz or halved to obtain the delta time for 208 Hz). This is useful because the data rate of the sensor is not guaranteed to be exactly the nominal value and, as a consequence, the delta time can deviate from the nominal value. The value in the ISPU_DTIME registers, properly rescaled for the configured data rate, can then be used when high accuracy is necessary, for example when integrating gyroscope data over time.

Just like the ISPU_DUMMY_CFG registers, both TIMESTAMP and ISPU_DTIME cannot be read from the ISPU in one step as 32-bit values, but must be read in two steps (first two bytes and last two bytes) and then recombined.

For example, the timestamp value can be read using the following code:

```
uint32_t timestamp = ((uint32_t)*((volatile uint16_t *)TIMESTAMP2) << 16) | *((volatile
uint16_t *)TIMESTAMP0);
uint64_t timestamp_us = timestamp * 25;
```

Note that the timestamp generation must be enabled by setting the TIMESTAMP_EN bit to 1 in the CTRL10_C register.

The delta time register value, instead, can be read and rescaled as follows:

```
union {
    uint32_t bytes;
    float value;
} dtime;

dtime.bytes = ((uint32_t)*((volatile uint16_t *)ISPU_DTIME_1) << 16) | *((volatile uint16_t
*)ISPU_DTIME_0);
dtime.value *= 104.0f / ISPU_ACTUAL_RATE;
```

Note that the delta time value contained in the ISPU_DTIME register is fixed and does not change at runtime. For this reason, the value may be read only once, for example during the initialization routine of an algorithm.

9.7 Outputs

In order to export the results of the processing, the ISPU can write data to be read from the device interface to a set of general-purpose registers. In particular, 32 16-bit registers are available, for a total of 64 bytes that can be used to encode any information. These registers are part of the ISPU interaction registers:

- ISPU_DOUT_00_L (10h / 6810h), ISPU_DOUT_00_H (11h / 6811h)
- ISPU_DOUT_01_L (12h / 6812h), ISPU_DOUT_01_H (13h / 6813h)
- ...
- ISPU_DOUT_31_L (4Eh / 684Eh), ISPU_DOUT_31_H (4Fh / 684Fh)

These registers are readable and writable from the ISPU, but are only readable by the external microcontroller from the device interface.

A block data update (BDU) mechanism in these registers is also available. Usually, if the output values are read synchronously to an interrupt signal generated by the ISPU when the outputs have just been written, this mechanism is not needed. However, if the output values are read asynchronously or if reading from the external microcontroller is very slow, it might happen that some bytes of a particular output value are read before being updated with a new value by the ISPU, while the other bytes are read after. The resulting read value is thus corrupted. The BDU prevents this problem by blocking the refresh of all the bytes of the value at the start of the read until all bytes have been read. Note that the write of the new value by the ISPU is successful, but the old value is sent over the device interface until all bytes have been read.

The block data update can be enabled by setting the BDU bit to 1 in the CTRL3_C register. In addition, since the ISPU output registers are general purpose and the data may be encoded in multiple ways, a few different configurations for the BDU are available to accommodate most usages. The available configurations are listed in Table 35 and can be set by writing the ISPU_BDU_[1:0] field in the CTRL9_C register.

Table 35. ISPU block data update configuration

ISPU_BDU_[1:0]	ISPU_DOUT_00_L to ISPU_DOUT_15_H	ISPU_DOUT_16_L to ISPU_DOUT_31_H
00	BDU disabled	BDU disabled
01	BDU on 2 bytes (16 outputs)	BDU on 4 bytes (8 outputs)
10	BDU on 2 bytes (16 outputs)	BDU on 2 bytes (16 outputs)
11	BDU on 4 bytes (8 outputs)	BDU on 4 bytes (8 outputs)

For example, by setting the ISPU_BDU_[1:0] field to 01, the first 32 bytes are considered to be 16 values of 2 bytes each, while the last 32 bytes are considered to be 8 values of 4 bytes each.

9.8 Algorithms

The ISPU is an embedded core that can be freely programmed to handle the processing of data as desired. However, the device also implements some mechanisms at a hardware level to facilitate the execution of multiple algorithms or processing logics and the management of their lifecycle.

The device is able to generate a number of interrupt requests (IRQ), as shown in Table 36. These should not be confused with the interrupts generated by the device on the INT1 and INT2 pins, described in Section 9.9 Interrupts.

Each interrupt request corresponds to a hardware signal that can be sent to the ISPU to execute a specific routine upon request. The user must create an interrupt vector table (IVT) to map each interrupt request to a proper routine to serve it. Each entry of the IVT must be implemented as a jump to the routine. The jump instruction must be placed at the address specified in Table 36. Note that the IVT must be placed at the beginning of the 32 KB RAM for code and read-only data, so the addresses specified in the table are relative to the address 0x10000.

Table 36. ISPU interrupt requests

Program address	Interrupt definition
0x00	Boot
0x04	Algorithm 00 run
0x08	Algorithm 01 run
0x0C	Algorithm 02 run
0x10	Algorithm 03 run
0x14	Algorithm 04 run
0x18	Algorithm 05 run
0x1C	Algorithm 06 run
0x20	Algorithm 07 run
0x24	Algorithm 08 run
0x28	Algorithm 09 run
0x2C	Algorithm 10 run
0x30	Algorithm 11 run
0x34	Algorithm 12 run
0x38	Algorithm 13 run
0x3C	Algorithm 14 run
0x40	Algorithm 15 run
0x44	Algorithm 16 run
0x48	Algorithm 17 run
0x4C	Algorithm 18 run
0x50	Algorithm 19 run
0x54	Algorithm 20 run
0x58	Algorithm 21 run
0x5C	Algorithm 22 run
0x60	Algorithm 23 run
0x64	Algorithm 24 run
0x68	Algorithm 25 run
0x6C	Algorithm 26 run
0x70	Algorithm 27 run
0x74	Algorithm 28 run

Program address	Interrupt definition
0x78	Algorithm 29 run
0x80	Algorithm 00 initialization
0x84	Algorithm 01 initialization
0x88	Algorithm 02 initialization
0x8C	Algorithm 03 initialization
0x90	Algorithm 04 initialization
0x94	Algorithm 05 initialization
0x98	Algorithm 06 initialization
0x9C	Algorithm 07 initialization
0xA0	Algorithm 08 initialization
0xA4	Algorithm 09 initialization
0xA8	Algorithm 10 initialization
0xAC	Algorithm 11 initialization
0xB0	Algorithm 12 initialization
0xB4	Algorithm 13 initialization
0xB8	Algorithm 14 initialization
0xBC	Algorithm 15 initialization
0xC0	Algorithm 16 initialization
0xC4	Algorithm 17 initialization
0xC8	Algorithm 18 initialization
0xCC	Algorithm 19 initialization
0xD0	Algorithm 20 initialization
0xD4	Algorithm 21 initialization
0xD8	Algorithm 22 initialization
0xDC	Algorithm 23 initialization
0xE0	Algorithm 24 initialization
0xE4	Algorithm 25 initialization
0xE8	Algorithm 26 initialization
0xEC	Algorithm 27 initialization
0xF0	Algorithm 28 initialization
0xF4	Algorithm 29 initialization

The interrupt request at 0x00 is generated when booting the ISPU (see Section 9.12 Boot procedure) and must jump to a procedure that executes the startup code and then puts the ISPU in the sleep state to wait for the first data sample from the sensor.

As indicated in Table 36, there is hardware support for up to 30 algorithms, from algorithm 00 to algorithm 29. The device is able to generate, for each algorithm, an interrupt request to execute the algorithm initialization code and an interrupt request to execute the run code of the algorithm.

The generation of the interrupt requests related to the algorithms is controlled by the user by writing, from the ISPU code, to the appropriate registers. First of all, the ISPU_GLB_CALL_EN bit in ISPU_GLB_CALL_EN (6800h) must be set to 1 in order to enable the interrupt request generation for all algorithms.

Then, in order to generate an interrupt request for an algorithm, the corresponding bit must be set to 1 in registers ISPU_CALL_EN_0 (68B8h), ISPU_CALL_EN_1 (68B9h), ISPU_CALL_EN_2 (68BAh) and ISPU_CALL_EN_3 (68BBh). Note that bit 0 of the ISPU_CALL_EN_0 register must be kept at 0. For example, in order to generate an interrupt request for algorithm 00, the bit ISPU_CALL_ALGO_0 (which is bit 1) of ISPU_CALL_EN_0 must be set to 1. When this bit is set to 1, the interrupt request for the corresponding algorithm is generated and the bit value remains at 1 until the interrupt service routine of the algorithm has completed. Once the routine has completed, the bit is automatically reset to 0. This allows monitoring the status of the execution of the algorithms and waiting for their completion before putting the ISPU in the sleep state to wait for the next data sample from the sensor.

When an interrupt is requested for an algorithm, the interrupt request generated normally corresponds to the routine for running the algorithm, except for the first time, when an interrupt request is generated to execute the algorithm initialization routine. In order to trigger again the execution of the algorithm initialization code, it is necessary to disable and re-enable the algorithm, as explained in the following paragraphs.

Of course the interrupt requests for the algorithms can be generated at any time, but it is recommended to generate them after the ISPU has been woken up due to a new data sample being available from the sensor. In this way the algorithms can be executed to process the new data sample and then the ISPU can be again put in the sleep state.

Figure 11 in Section 9.9 Interrupts shows the usage of the ISPU_CALL_EN registers in the recommended processing cycle.

If the execution of multiple algorithms is requested at the same time, the algorithm routines are executed in order, from algorithm 29 to algorithm 00.

A set of registers, part of the ISPU interaction registers, is available to enable or disable each algorithm from the device interface:

- ISPU_ALGO0 (70h / 6870h) allows enabling / disabling algorithms 00 to 07.
- ISPU_ALGO1 (71h / 6871h) allows enabling / disabling algorithms 08 to 15.
- ISPU_ALGO2 (72h / 6872h) allows enabling / disabling algorithms 16 to 23.
- ISPU_ALGO3 (73h / 6873h) allows enabling / disabling algorithms 24 to 29.

As mentioned above, disabling and re-enabling an algorithm using these registers resets an internal flag that causes the algorithm initialization routine to be called the next time an interrupt request is generated. Aside from this, it is up to the user to use these registers within the ISPU code to enable and disable the execution of the algorithms, which exclusively depends on the value of the ISPU_GLB_CALL_EN bit and on which bits are written to 1 in the ISPU_CALL_EN registers. For example, when the ISPU is woken up, the ISPU_ALGO registers can be read in order to determine which algorithms are enabled and set the corresponding bits in the ISPU_CALL_EN registers, thus executing only the enabled algorithms. This can be achieved as follows:

```
*((volatile uint32_t *)ISPU_CALL_EN) = *((volatile uint32_t *)ISPU_ALGO) << 1;
```

The bits in the ISPU_ALGO registers can also be cleared from the ISPU in order to disable the algorithms. This can be achieved by writing 1 to the bit corresponding to the algorithm to disable. While this might be counterintuitive, writing 1 means requesting the bit to be cleared. However, it is not possible to enable the algorithms from the ISPU.

9.9 Interrupts

In order to keep the external microcontroller sleeping most of the time, it is necessary to avoid polling for new results from the ISPU. For this reason, it is possible to generate interrupt signals on the INT1 and INT2 pins of the device to wake up the external microcontroller only when necessary.

First of all, routing the interrupts must be enabled:

- Setting INT1_ISPU to 1 in the MD1_CFG register enables routing the interrupts to the INT1 pin.
- Setting INT2_ISPU to 1 in the MD2_CFG register enables routing the interrupts to the INT2 pin.

The interrupts can then be configured in two different modes by setting the LATCH bit in the ISPU_CONFIG register, which is part of the ISPU interaction registers:

- Pulsed mode is selected by setting the LATCH bit to 0 (default value).
- Latched mode is selected by setting the LATCH bit to 1.

If the interrupts are configured in pulsed mode, the interrupt pins must be directly controlled from the ISPU code by writing 0 or 1 to the INT1 and INT2 bits in register ISPU_INT_PIN (685Ch). Writing 0 causes the pin to be set to the inactive level, writing 1 causes the pin to be set to the active level.

Note that the active level of the interrupt pins can be either set to high or low depending on the value of the H_LACTIVE bit in the CTRL3_C register. If the H_LACTIVE bit is set to 0, the interrupt pins are active high, otherwise they are active low.

In order to generate a pulsed interrupt signal, the INT1 or INT2 bit (based on the desired pin) can be set to 1 at the end of the processing of a data sample (if an interrupt must be raised) and set back to 0 when the ISPU is woken up because the next data sample is available. In this way, the interrupt line remains active for no longer than the time between the generation of two consecutive samples and the external microcontroller does not need to perform any action to reset the line.

If the interrupts are instead configured in latched mode, the ISPU is unable to directly set the pin level. Instead, the interrupt line is automatically controlled by the device.

A latched interrupt is generated when one or multiple interrupt flags are raised in the interrupt status registers, which are part of the ISPU interaction registers:

- ISPU_INT_STATUS0 (58h / 6858h) contains the interrupt flags for algorithms 00 to 07.
- ISPU_INT_STATUS1 (59h / 6859h) contains the interrupt flags for algorithms 08 to 15.
- ISPU_INT_STATUS2 (5Ah / 685Ah) contains the interrupt flags for algorithms 16 to 23.
- ISPU_INT_STATUS3 (5Bh / 685Bh) contains the interrupt flags for algorithms 24 to 29.

In addition, the bits in the ISPU_INT1_CTRL or ISPU_INT2_CTRL registers corresponding to the raised interrupt flags must be set to 1, otherwise, the flags are ignored and the latched interrupt is not generated on the pins. These control registers, part of the ISPU interaction registers, must be written from the device interface and can be used to enable or disable routing the interrupts to the INT1 and INT2 pins for each algorithm:

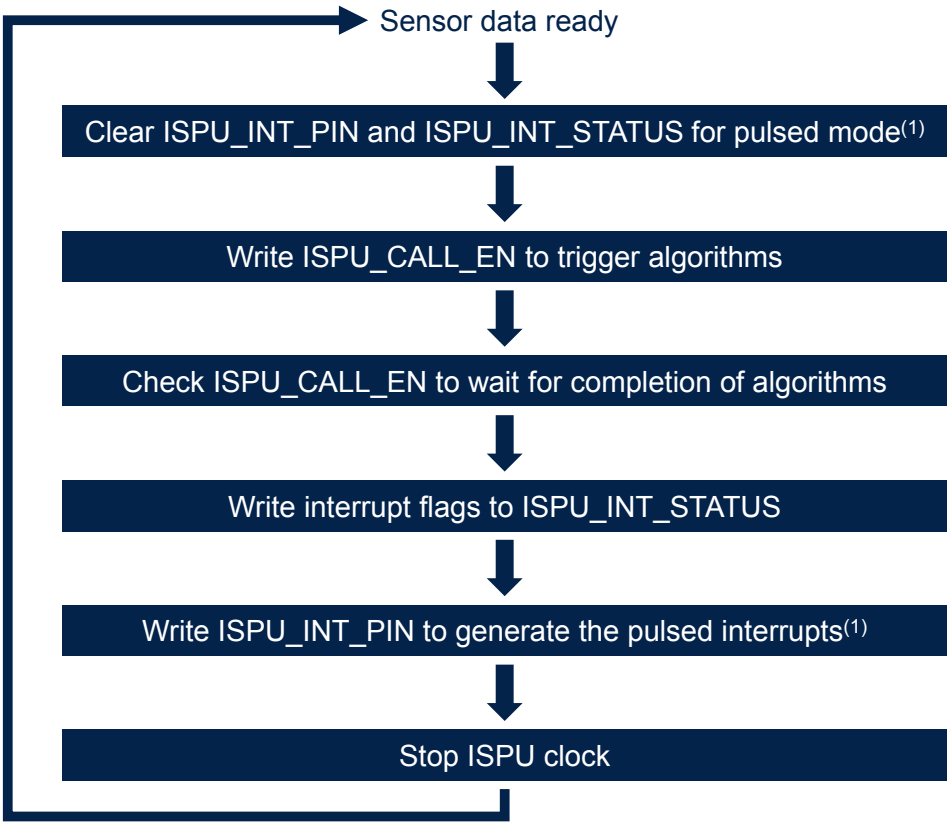
- ISPU_INT1_CTRL0 (50h / 6850h) allows enabling / disabling routing to INT1 for algorithms 00 to 07.
- ISPU_INT1_CTRL1 (51h / 6851h) allows enabling / disabling routing to INT1 for algorithms 08 to 15.
- ISPU_INT1_CTRL2 (52h / 6852h) allows enabling / disabling routing to INT1 for algorithms 16 to 23.
- ISPU_INT1_CTRL3 (53h / 6853h) allows enabling / disabling routing to INT1 for algorithms 24 to 29.
- ISPU_INT2_CTRL0 (54h / 6854h) allows enabling / disabling routing to INT2 for algorithms 00 to 07.
- ISPU_INT2_CTRL1 (55h / 6855h) allows enabling / disabling routing to INT2 for algorithms 08 to 15.
- ISPU_INT2_CTRL2 (56h / 6856h) allows enabling / disabling routing to INT2 for algorithms 16 to 23.
- ISPU_INT2_CTRL3 (57h / 6857h) allows enabling / disabling routing to INT2 for algorithms 24 to 29.

In latched mode, the interrupt line is set back to the inactive level when the external microcontroller reads the raised bits in the interrupt status registers.

Note that the interrupt flags in the interrupt status registers are not automatically generated, but it is up to the user to write, from the ISPU code, the corresponding bit to 1 when an algorithm triggers an interrupt (in pulsed mode, this should be done before generating the actual interrupt signal on the pin). If latched mode is enabled, then the bits are automatically reset when reading the interrupt status registers from the device interface. If latched mode is disabled, it is once again up to the user to set the bits back to 0. The reset of these bits should be performed at the same time as the the ISPU_INT_PIN register is written in order to set the interrupt pin back to the inactive level (for example, when the ISPU is woken up because the next data sample is available). The purpose of the interrupt status registers is to allow the external microcontroller to understand which algorithms have generated the interrupt.

Figure 11 shows the usage of the ISPU_INT_PIN and ISPU_INT_STATUS registers in the recommended processing cycle. Note that, as explained above, writing the ISPU_INT_PIN register does not have an effect if latched mode is enabled, but should be done anyway to support pulsed mode with the same code.

Figure 11. ISPU recommended processing cycle



1. This step is not necessary if interrupt latched mode is enabled.

In the main register page, a copy of the interrupt flags is also available in registers ISPU_INT_STATUS0_MAINPAGE, ISPU_INT_STATUS1_MAINPAGE, ISPU_INT_STATUS2_MAINPAGE, and ISPU_INT_STATUS3_MAINPAGE, which are readable without having to enable the access to the ISPU interaction registers. If latched mode is enabled, in order to clear the interrupt, reading these registers is equivalent to reading the ISPU_INT_STATUS registers.

Of course the bits of the interrupt status registers can be used freely, so if the concept of multiple algorithms is not used, the bits can be used, for example, to distinguish different types of interrupts related to the same algorithm.

Note that, in pulsed mode, the ISPU_INT1_CTRL and ISPU_INT2_CTRL registers produce no effect on the device, and it is up to the user to use them within the ISPU code to generate an interrupt only if the bit corresponding to the algorithm that generates the interrupt is set to 1, meaning that the interrupt must be routed to the pin. A generic interrupt can be generated at software level and then, based on the ISPU_INT1_CRTL and ISPU_INT2_CTRL register values, it can be decided whether to generate an interrupt on the desired interrupt pin. This can be achieved with some simple masking, as shown in the following code.

```
*((volatile uint8_t *)ISPU_INT_PIN) = (((int_status & *((volatile uint32_t *)ISPU_INT1_CTRL)
> 0) << 0) | (((int_status & *((volatile uint32_t *)ISPU_INT2_CTRL) > 0) << 1);
```

In the snippet of code above, the *int_status* variable contains the interrupt flags for all algorithms (this is also the value that can be written to the interrupt status registers). The value of this variable is masked with the content of the interrupt control registers to determine whether to actually trigger an interrupt on each of the two pins (INT1 and INT2).

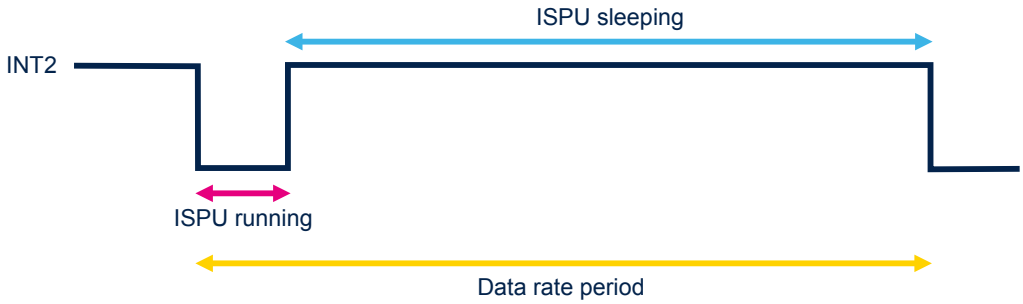
In addition to the physical interrupts, an additional register, ISPU_S2IF_FLAG (0Eh-0Fh / 680Eh-680Fh), provides bits that can be set by the ISPU and cleared from the device interface by the external microcontroller. This register is part of the ISPU interaction registers.

The intended use of the ISPU_S2IF_FLAG register is to raise flags to the external microcontroller, which can check the register, detect the bits that are set to 1, perform specific actions accordingly and then clear the bits to signal that the request received from the ISPU was served. In this sense, these bits can be used to implement a sort of software interrupt. Of course, in this case, the external microcontroller needs to poll the register to detect when the flags are raised. Alternatively, this register could be used in combination with a physical interrupt to ask the external microcontroller to perform some actions when the interrupt is received.

In order to set a bit of the ISPU_S2IF_FLAG register, it is sufficient to write 1 to it from the ISPU. In order to clear the bit, the bit must be written again to 1 from the device interface. While this might be counterintuitive, writing 1 means requesting the bit to be cleared.

The INT2 pin can also be used for monitoring the active time of the ISPU. The ISPU sleep signal can be routed to the INT2 pin of the device by setting the INT2_SLEEP_ISPU bit to 1 in the INT2_CTRL register. When this functionality is enabled, the INT2 line is set to the inactive level while the ISPU is running, and set to the active level while the ISPU is sleeping. Using the signal produced on the INT2 pin, it is then possible to measure the total execution time of processing for one data sample. This can be achieved by measuring the time that the signal remains continuously at the inactive level, for example using a logic analyzer or a timer on the external microcontroller. Figure 12 shows an example of the signal produced on the INT2 pin after enabling the INT2_SLEEP_ISPU bit.

Figure 12. Example of the ISPU sleep signal on the INT2 pin (H_LACTIVE = 0)



As explained in Section 9.1 Processing cycle and rate configuration, evaluating the execution time is important to make sure that the processing is completed before the next data sample is available. Additionally, it makes it possible to evaluate the impact of optimizations to improve the performance of the processing logic, which, even if it is already fast enough to process data at the configured data rate, can still be improved to reduce the active time of the ISPU and thus the power consumption.

Another way to evaluate the performance of the implemented code is to set the interrupts in pulsed mode and control either or both interrupt pins directly. This allows measuring the performance of specific portions of code instead of the whole processing logic. This can be achieved, for example, by setting an interrupt pin to the inactive level just before the portion of code of interest and setting it back to the active level just after. The execution time for the portion of code of interest can then be obtained by measuring the time that the interrupt signal remains continuously at the inactive level (using a logic analyzer or a timer on the external microcontroller).

9.10 Memory access

The ISPU is supported by two separate RAMs, one of 32 KB size for code and read-only data, and one of 8 KB size for variable data.

In addition to being accessible from the ISPU, the contents of the two RAMs can also be accessed from the device interface (I²C or SPI) using specific procedures. As explained in Section 9.11 Program loading, this is necessary in order to load the binary of the program to the RAM.

In order to write to the RAM, these steps must be followed:

1. The MEM_SEL bit in the ISPU_MEM_SEL register must be configured to select one of the two RAMs.
   a. Writing 0 to MEM_SEL selects the variable data RAM (8 KB).
   b. Writing 1 to MEM_SEL selects the code and read-only data RAM (32 KB).
2. Write 0 to READ_MEM_EN in the ISPU_MEM_SEL register to enable writing.
3. Write the address that the data must be written to in the ISPU_MEM_ADDR registers.
   a. The ISPU_MEM_ADDR1 register must contain the most significant byte of the address.
   b. The ISPU_MEM_ADDR0 register must contain the least significant byte of the address.
4. Write the data at the selected location by writing it in the ISPU_MEM_DATA register. If multiple writes are performed without setting a new address, each new byte is written to the next location (the address is automatically incremented).

Note that the autoincrement of the address does not work when crossing the following memory locations in the code and read-only RAM (32 KB):

- 1FFFh → 2000h
- 3FFFh → 4000h
- 5FFFh → 6000h

When crossing the above locations, the address must be explicitly set in the ISPU_MEM_ADDR registers.

In order to read from the RAM, these steps must be followed:

1. The MEM_SEL bit in the ISPU_MEM_SEL register must be configured to select one of the two RAMs.
   a. Writing 0 to MEM_SEL selects the variable data RAM (8 KB).
   b. Writing 1 to MEM_SEL selects the code and read-only data RAM (32 KB).
2. Write 1 to READ_MEM_EN in the ISPU_MEM_SEL register to enable reading.
3. Write the address that the data must be read from in the ISPU_MEM_ADDR registers.
   a. The ISPU_MEM_ADDR1 register must contain the most significant byte of the address.
   b. The ISPU_MEM_ADDR0 register must contain the least significant byte of the address.
4. Read the data at the selected location by reading the ISPU_MEM_DATA register. If multiple reads are performed without setting a new address, each new byte is read from the next location (the address is automatically incremented).

Note that every time a new address is set in the ISPU_MEM_ADDR registers, the first read byte must be discarded.

Note also that all the registers mentioned above are part of the ISPU interaction registers, so, in order to perform the above procedures, access to that set of registers must be enabled as explained in Section 9.3 ISPU interaction registers.

The above procedures should be mainly used to load the ISPU program and optionally read it back to check if it was correctly loaded. It is possible to access the RAM at runtime (after the ISPU boot), but it is not recommended since it must be guaranteed that the ISPU is in sleep mode (clock stopped) while accessing the memory.

9.11 Program loading

Once a program has been compiled using the ISPU toolchain, it must be loaded to the RAM of the device before performing the boot procedure described in Section 9.12 Boot procedure.

Before loading the program to the device, it is recommended to always perform a software reset of the ISPU by setting the SW_RESET_ISPU bit of the FUNC_CFG_ACCESS register to 1 and then immediately setting it back to 0. This step is necessary if the ISPU was previously booted. It resets the ISPU core and registers (ISPU interaction registers and ISPU functions registers).

The primary output of the toolchain is a binary ELF (executable and linkable format) file containing the binary of the program. However, it is possible to convert the ELF file to the Motorola S-record (SREC) format, which is text-based and much easier to parse. This conversion can be done using the following command:

```
reisc-objcopy -O srec ispu.elf ispu.srec
```

The SREC file contains different types of lines, but the lines of interest are those starting with "S2". These lines provide the actual data to write to the device and have the following format:

	type	count	address	data	checksum
S	2	N N	A A A A A A	D[0] D[0] D[1] D[1] ... D[N-1] D[N-1]	C C

- **S** is the first character for every line in an SREC file.
- The **type** field indicates the record type, in this case (type = 2) a data record with a 24-bit address.
- The **count** field is the number of bytes in the record (including **address**, **data**, and **checksum**), expressed as one byte in hexadecimal format.
- The **address** field is expressed as a 24-bit value in hexadecimal format and represents the address the **data** must be written to. Addresses for the ISPU utilize only 17 of the 24 bits, the remaining bits are set to zero. The most significant bit (of the 17 bits) represents the RAM selection and can be mapped to the MEM_SEL bit in the ISPU_MEM_SEL register. The remaining 16 bits represent the address that can be set in the ISPU_MEM_ADDR1 (most significant byte) and ISPU_MEM_ADDR0 (least significant byte) registers.
- The **data** field is also expressed in hexadecimal format and can vary in size depending on the **count** field. These bytes must be loaded sequentially to the device RAM starting from the **address**.
- The **checksum** field contains a one byte checksum of the **count**, **address**, and **data** fields, expressed in hexadecimal format.

In order to load the program it is then possible to parse the SREC file and, for each "S2" line, select the RAM to write to, set the address specified, and write the data bytes included in the line, all while following the write procedure described in Section 9.10 Memory access.

Note that the address automatic increment can be leveraged to reduce the number of writes to the device, but the address must be explicitly set when writing non-consecutive bytes or when crossing the locations that inhibit the automatic increment listed in Section 9.10 Memory access.

Note: The toolchain provided by STMicroelectronics allows converting the SREC format into ready-to-use formats that are supported by STMicroelectronics tools and that can be directly integrated in custom projects. These formats contain the sequence of write operations to load the program from the device interface.

9.12 Boot procedure

After the program has been loaded to the device RAM, the following procedure must be followed in order to boot the ISPU core:

1. Configure the ISPU clock to 5 MHz by setting the ISPU_CLK_SEL bit in the CTRL10_C register to 0.
2. Set the ISPU IRQ rate to 0 Hz by writing ISPU_RATE_[3:0] in the CTRL9_C register to 0h.
3. Power on the accelerometer sensor (if not already on) by writing ODR_XL[3:0] in the CTRL1_XL register to a value other than 0h.
4. Enable the ISPU clock by writing the CLK_DIS bit of the ISPU_CONFIG register to 0.
5. Disable the ISPU reset by writing the ISPU_RST_N bit of the ISPU_CONFIG register to 1.
6. Wait for the end of the boot before any further sensor configuration.

Note that steps 4 and 5 can be performed with one single write to the ISPU_CONFIG register. ISPU_CONFIG is part of the ISPU interaction registers.

The boot time depends on the loaded program, so, in order to know if the boot has ended, the external microcontroller can poll the BOOT_END bit in the ISPU_STATUS (04h / 6804h) register and wait for its value to become 1 (boot finished). The ISPU_STATUS register is part of the ISPU interaction registers. The boot is to be considered finished when the boot code has been executed and the program is ready to process data samples. Since the boot code is written by the user and can change, it is up to the user to set the BOOT_END bit to 1 from the ISPU code once the boot code has been successfully executed.

As an alternative to polling the BOOT_END bit, the external microcontroller can implement a delay to wait for a predefined time. However, in order to define the value of this delay, the boot time should be estimated in advance during the development phase. For example, using the templates and examples provided by STMicroelectronics, a wait of 5 ms is sufficient. The BOOT_END bit must be in any case set to 1 from the ISPU, otherwise the ISPU does not go to sleep when trying to stop the clock as explained in Section 9.1 Processing cycle and rate configuration.

Note: If a reboot of the ISPU is needed, the software reset described in Section 9.11 Program loading must be performed before the boot procedure, even if no new program needs to be loaded.

The boot procedure can be automatically generated by the toolchain provided by STMicroelectronics.

9.13 Clock configuration and performance

The clock of the ISPU core can be configured to two different settings:

- 5 MHz (default) by setting the ISPU_CLK_SEL bit in the CTRL10_C register to 0.
- 10 MHz by setting the ISPU_CLK_SEL bit in the CTRL10_C register to 1.

If the gyroscope is powered on (the accelerometer may be powered on or off), the ISPU at 10 MHz consumes, when running, roughly double the power than it would at 5 MHz (see Section 9.14 Power consumption). However, the processing logic is executed in about half the time due to the doubled frequency and, as a consequence, the ISPU is kept running for half the time. Since the ISPU consumes double the power but for half the time, the average power consumption remains roughly the same. If only the accelerometer is powered on, setting the clock to 10 MHz causes an additional fixed contribution to the power consumption, which means that the ISPU at 10 MHz consumes more than double the power than the ISPU at 5 MHz.

Based on the considerations above, the 10 MHz clock, especially if the gyroscope is powered off, should be used only if at 5 MHz the processing logic is not able to complete before the next sample arrives, as explained in Section 9.1 Processing cycle and rate configuration.

However, the clock configuration is not the only way to improve the performance of the processing logic. It may be possible to optimize the code, also considering the specific characteristics of the ISPU architecture. This approach should be preferred over increasing the clock frequency if using only accelerometer in order to avoid higher power consumption and should anyway be considered in order to reduce the execution time of the processing logic and thus the overall power consumption of the device. A few suggestions for code optimization in the ISPU are to:

- avoid floating-point divisions whenever possible (for example, divisions using constants may be done by multiplications using the inverse value)
- avoid, if possible, the usage of complex mathematical functions like square root (for example, if computing the norm of a vector, using the squared norm instead may be acceptable depending on the algorithm) or, if necessary, use approximations that are faster to compute than the standard library functions
- avoid too many memory accesses (for example, use ring buffers instead of linear buffers that would require shifting all elements each time a new element is inserted)

9.14 Power consumption

As explained in Section 9.13 Clock configuration and performance, the ISPU clock can be set to either 5 MHz or 10 MHz. Independently from the clock setting, at any time, the ISPU can be in one of the following two states:

- Sleep (clock stopped): this is the ISPU state if the clock is disabled (CLK_DIS bit set to 1 in the ISPU_CONFIG register) or if the clock was stopped to put the ISPU in the sleep state until the next data sample is available (see Section 9.1 Processing cycle and rate configuration).
- Run (clock running): this is the ISPU state if the clock is enabled (CLK_DIS bit set to 0 in the ISPU_CONFIG register) and the ISPU is currently awake to process a data sample.

Table 37 shows, for each clock setting, the typical current consumption in the two states of the ISPU. Note that these current consumption numbers must be added to the ones listed in Table 9 in Section 3 Operating modes (related to the sensor only) in order to obtain the overall current consumption of the device.

Table 37. ISPU current consumption (@ Vdd = 1.8 V, T = 25 °C)

Clock	Sleep	Run
5 MHz	0 µA(1)	1150 µA(2)
10 MHz	0 µA(1)	2300 µA(2)

1. *The current consumption of the ISPU in the sleep state (in the order of a few microamperes) is always present since the device boot. For this reason, it cannot be distinguished from the sensor current consumption and it is included in the values listed in Table 9.*
2. *Typical current consumption when configuring the ISPU with the worst-case load. The value is specified by design, not tested in production and not guaranteed. In typical applications, the ISPU very seldom reaches the worst-load condition, thus the power consumption is usually far lower than this figure of merit.*

Note that the 10 MHz clock requires powering on an internal block that is also used when the gyroscope is powered on. For this reason, if the gyroscope is in power-down mode, configuring the ISPU clock to 10 MHz adds about 32 µA (in the typical case) to the values listed in Table 37 (for both sleep and run states).

Also note that if the boot procedure of the ISPU has not been performed, the ISPU adds no current consumption (even if the clock is configured to 10 MHz) and the values listed in Table 9 represent the overall current consumption of the device.

Normally, in an application, the ISPU is not always running, but it is running while processing the current data sample and then sleeping until the next data sample is available. The actual current consumption of the ISPU is then determined by the ratio between the time the ISPU is in the run state and the time it is in the sleep state.

Note that the values listed in Table 37 are the typical values measured with the worst-case load. Based on the actual code running on the ISPU (for example the types of operations, reads and writes to the RAM, reads and writes to the registers), the current consumption could be (and usually is) significantly lower.

As an example, a 6-axis sensor fusion algorithm using accelerometer and gyroscope data, running at 104 Hz with the ISPU clock configured at 5 MHz, is executed in 3.2 ms and consumes 226 µA. That is the average current consumption considering both when the ISPU is in the run state and when it is in the sleep state. In this case, while the ISPU is in the run state, it consumes 677 µA.

An anomaly detection algorithm based on the features computation of the accelerometer data, running at 416 Hz with the ISPU clock configured at 5 MHz, detecting anomalies with respect to one class, is executed in 251 µs and consumes 76 µA. That is the average current consumption considering both when the ISPU is in the run state and when it is in the sleep state. In this case, while the ISPU is in the run state, it consumes 729 µA.

Appendix A

Table 38. ISPU instruction set

Group	Acronym	Description
Move	LD	Load data from memory
	ST	Store data in memory
	MOV	Move data between registers
Arithmetic	UEXT	Unsigned-value extension
	SEXT	Signed-value extension
	ADD	Integer addition
	SUB	Integer subtraction
	MUL	Integer multiplication
Floating	FADD	Floating-point single-precision addition
	FSUB	Floating-point single-precision subtraction
	FMUL	Floating-point single-precision multiplication
Shift	SRL	Shift-right logical
	SRA	Shift-right arithmetic
	SLA	Shift-left arithmetic
	ROTC	Rotate with carry
	ROT	Rotate left or right
Logical	AND	Bitwise AND
	OR	Bitwise OR
	XOR	Bitwise XOR
	CMP	Compare
Branch	JPD	Jump displacement
	JPI	Jump from immediate
	JPR	Jump from register
	JLR	Jump from register and link
	JLI	Jump from immediate and link
	RET	Return
	RFE	Return from exception
Special	NOP	No operation

Revision history

Table 39. **Document revision history**

Date	Version	Changes
04-Aug-2022	1	Initial release

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

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

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

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

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

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

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

© 2022 STMicroelectronics – All rights reserved