Lab1 ~ Lab5 共用規定

- 上課時間: 14:10~17:00；地點 @新館一樓 65104
- 每一個 lab 最晚 都會在上課當天中午12:00前上傳投影片到 moodle，
  為避免教室網路訊號不好，請同學在14:00上課前先下載投影片至電腦中。
- 每一個 lab 佔總分 8%, 獨立計分.　(Final Project 佔總分 60%)
- Lab 完成後, 要在 7 天內寫好 lab report 上傳 moodle。
- 若 lab 下課前有做完，我們會現場幫你評分。
- 若 lab 下課前沒做完，會有補交機制 (各 lab 規定方式可能不同)，
  期限內有完成就不會扣分 (期限為 7 天內，超過不計分)。

Lab3 規定

- Lab3 補交機制 (各 lab 規定方式可能不同)

  若未能於當天下課前完成(或是你沒筆電)，請在 4/24 23:59 之前，將 lab 完成的結果錄影後，上傳到自己的 google 雲端， 然後將影片播放連結 寄信給 Lab3 負責助教 (下一頁有mail)， 助教評分後，會回覆你的信件。

- 不論是現場完成，或是寄影片連結，都要寫 lab report (上傳moodle)。

LAB 3

E-mail : NN6131037@gs.ncku.edu.tw

Date: 2025/04/24

OUTLINE

01 Prelab

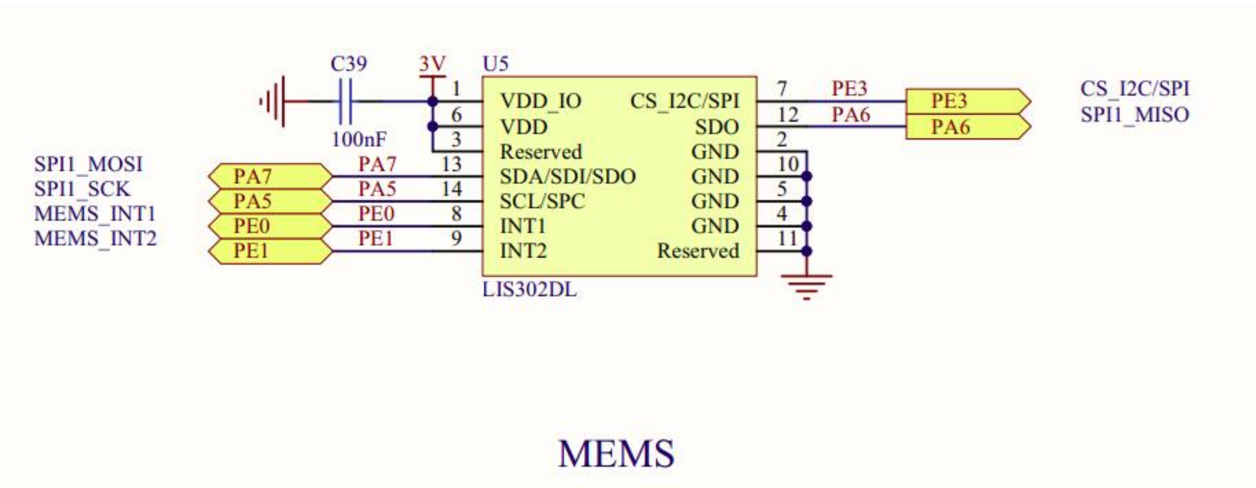
02 Lab 3 requirement

03 Semaphore introduction

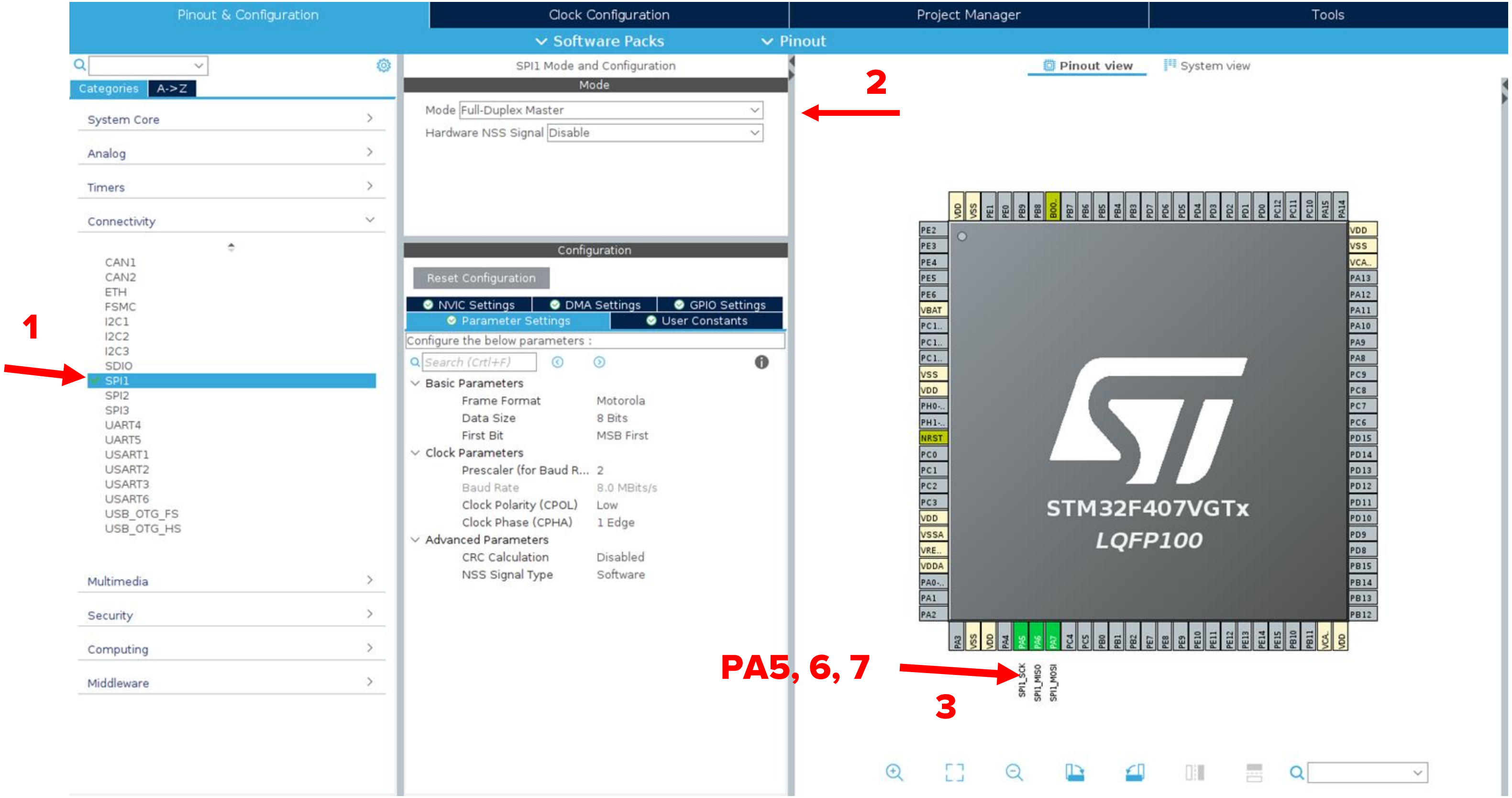
04 Tools introduction

Motion sensor

- Lab預設motion sensor型號: ST MEMS LIS3DSH
- The STM32F407VG microcontroller controls this motion sensor through the SPI interface.
- 下圖為 加速規 接在開發版的哪些pin腳上。

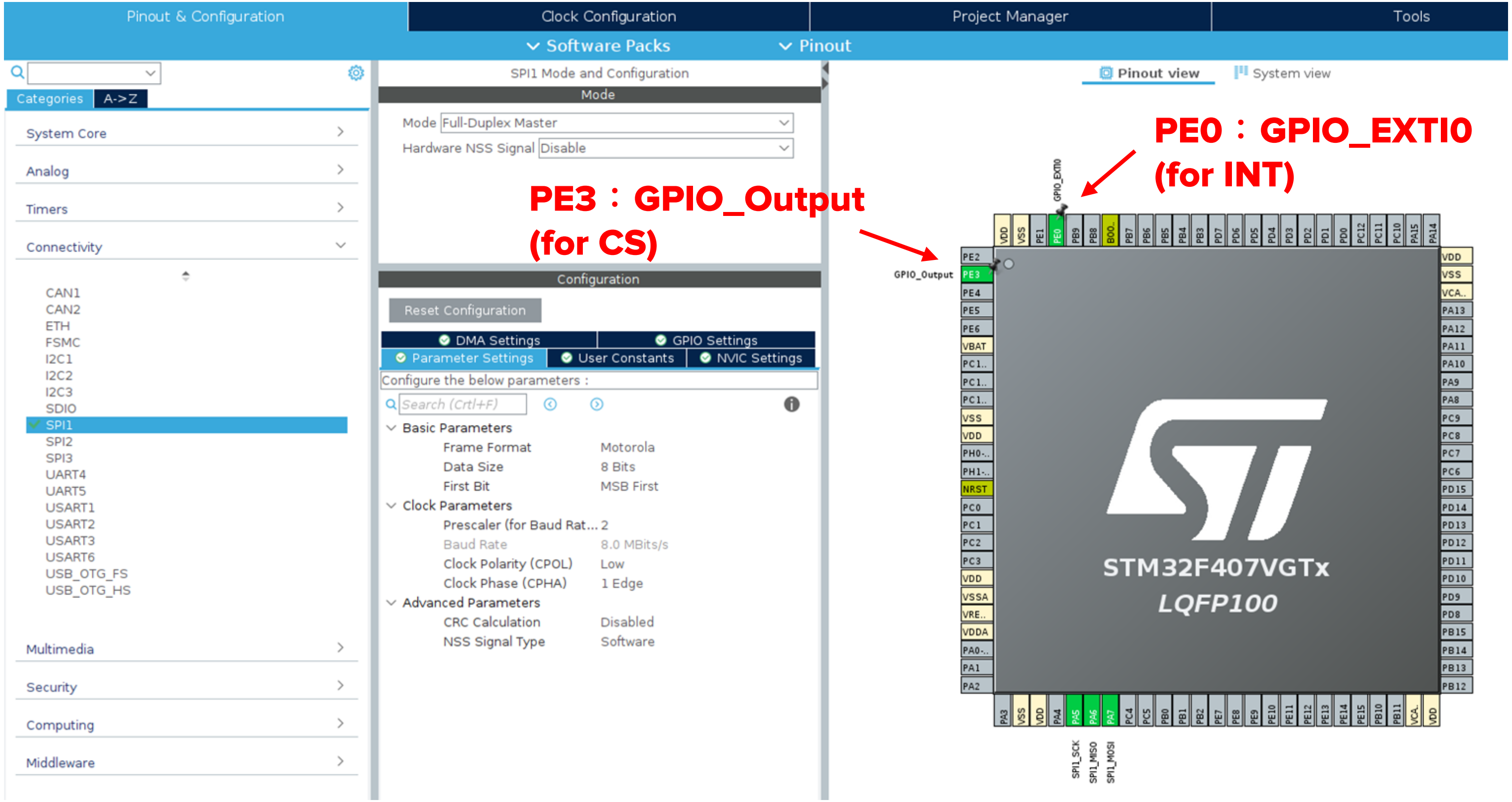


Motion sensor setup

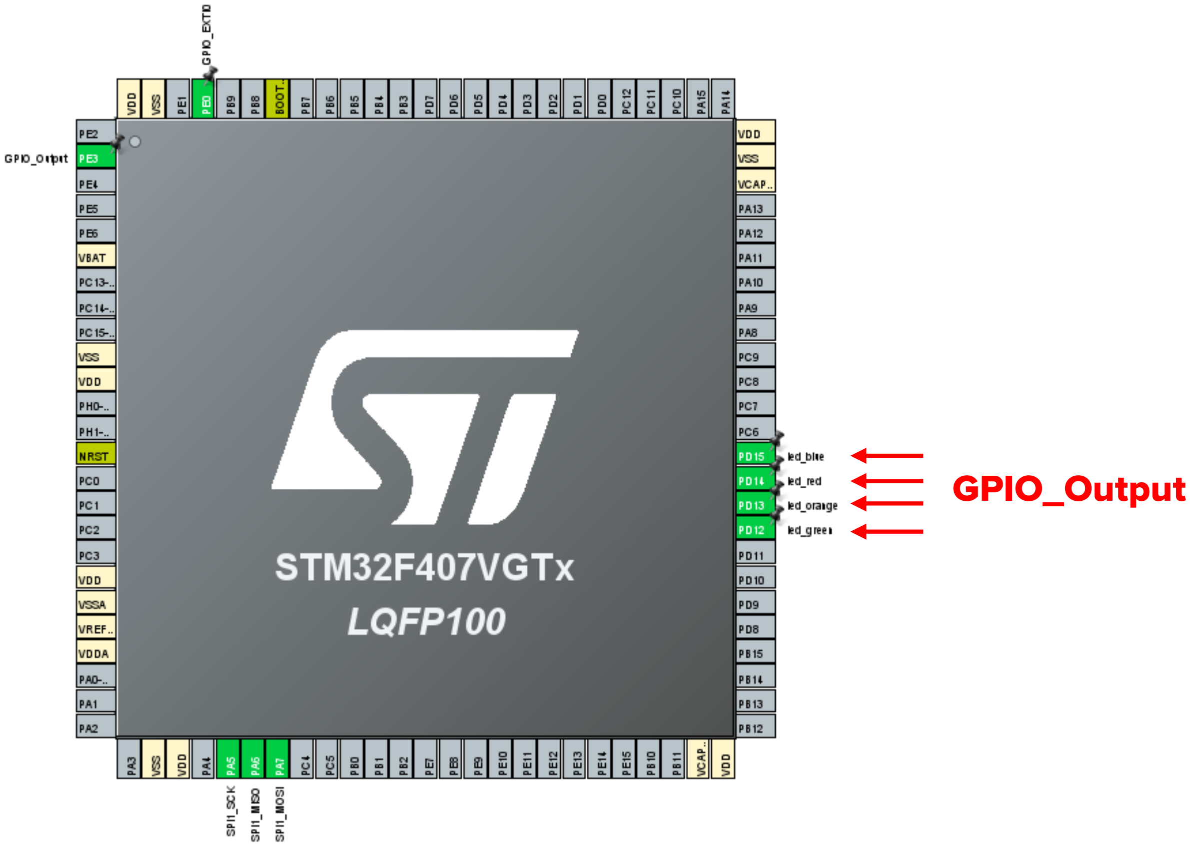


Motion sensor setup

(承接lab0，記得設定 綠 LED 燈相關 pin 腳)



承接lab0，記得設定 LED 燈相關 pin 腳



MEMS_Read/ MEMS_Write

- 透過SPI來讀寫register 的函式。

```
/* USER CODE BEGIN 0 */
void MEMS_Write(uint8_t address,uint8_t data){
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi1,&address,1,10);
    HAL_SPI_Transmit(&hspi1,&data,1,10);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3,GPIO_PIN_SET);
}
void MEMS_Read(uint8_t address,uint8_t *data){
    address |= 0x80;
    HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi1,&address,1,10);
    HAL_SPI_Receive(&hspi1,data,1,10);
    HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
}
/* USER CODE END 0 */
```

Motion sensor register

- 這些是能用來控制加速規的一些register，事先 define 好每個 register 的 address 方便之後使用。

```c
/* USER CODE BEGIN PM */
#define LIS3DSH_WHO_AM_I_ADDR             0x0F
#define LIS3DSH_STAT_ADDR                 0x18
#define LIS3DSH_CTRL_REG4_ADDR            0x20
#define LIS3DSH_CTRL_REG1_ADDR            0x21
#define LIS3DSH_CTRL_REG2_ADDR            0x22
#define LIS3DSH_CTRL_REG3_ADDR            0x23
#define LIS3DSH_CTRL_REG5_ADDR            0x24
#define LIS3DSH_CTRL_REG6_ADDR            0x25

#define LIS3DSH_STATUS_ADDR               0x27

#define LIS3DSH_OUT_X_L_ADDR              0x28
#define LIS3DSH_OUT_X_H_ADDR              0x29
#define LIS3DSH_OUT_Y_L_ADDR              0x2A
#define LIS3DSH_OUT_Y_H_ADDR              0x2B
#define LIS3DSH_OUT_Z_L_ADDR              0x2C
#define LIS3DSH_OUT_Z_H_ADDR              0x2D

#define LIS3DSH_ST1_1_ADDR                0x40
#define LIS3DSH_ST1_2_ADDR                0x41
#define LIS3DSH_THRS1_1_ADDR              0x57
#define LIS3DSH_MASK1_B_ADDR              0x59
#define LIS3DSH_MASK1_A_ADDR              0x5A
#define LIS3DSH_SETT1_ADDR                0x5B
```

Motion sensor testing

預設值0x3F

7.4 WHO_AM_I (0Fh)

Who_AM_I register.

Table 20. WHO_AM_I register default values

0	0	1	1	1	1	1	1

- 透過讀取 WHO_AM_I register 的值，來使綠燈閃爍。

```
void LED_Task(void *pvParameter)
{
    for(;;){
        uint8_t data;
        MEMS_Read(LIS3DSH_WHO_AM_I_ADDR,&data);
        if(data == 0x3F){
          HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
        }
        vTaskDelay(500/portTICK_RATE_MS);
    }

}
```

測試code連結: Code

理論上如果沒有問題，板子的綠色 LED燈會閃爍

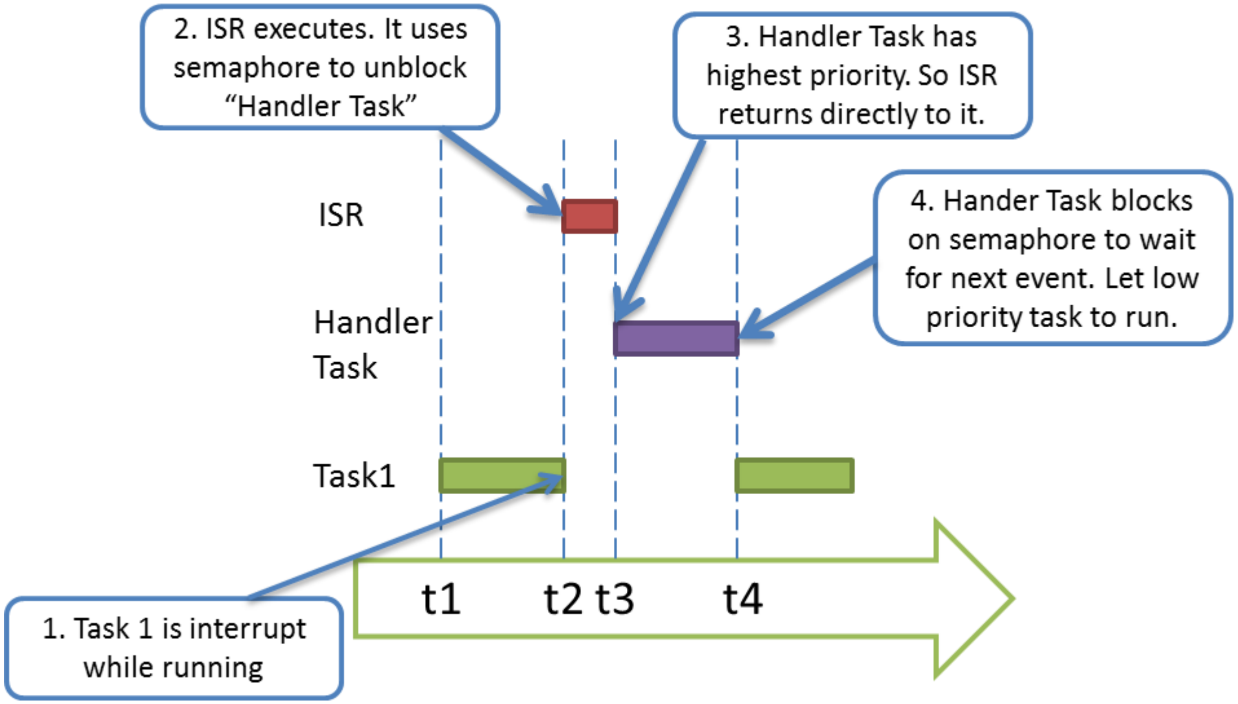
OUTLINE

Lab 3 Introduction

LED	green	orange	red
PIN	PD12	PD13	PD14

- Must use sensor interrupt : motion detection.
  - When you shake your board, it will trigger the interrupt.
- Please use the deferred interrupt handling task.
  - Use semaphore.
  - ISR will give the semaphore and the handler task enters the running state.
- At the beginning, the green LED blinking, then shake the board, the red LED triggered（switch state） by ISR and the orange LED blinking five times in handler task.
- When orange LED blinking, you should not trigger the sensor interrupt if you shake the board.

Deferred interrupt handling task



Lab3 demo

lab 3 demo

Lab3 grading (總成績 8 %)

- (3%) 正確觸發interrupt，且執行ISR(即搖晃板子有用)
- (1%) 亮燈順序正確
- (1%) 不會無緣無故解開鎖執行handler task，且在handler task執行完之前無法觸發 interrupt。
- (3%)Lab report (一定要交)

OUTLINE

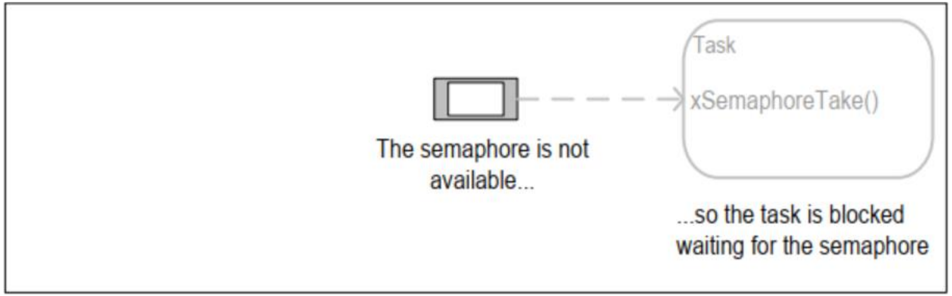
FreeRTOS Semaphore

- Three types of semaphores
  - Binary
  - Counting
  - Mutex

- Often used to control access to shared resources and synchronization

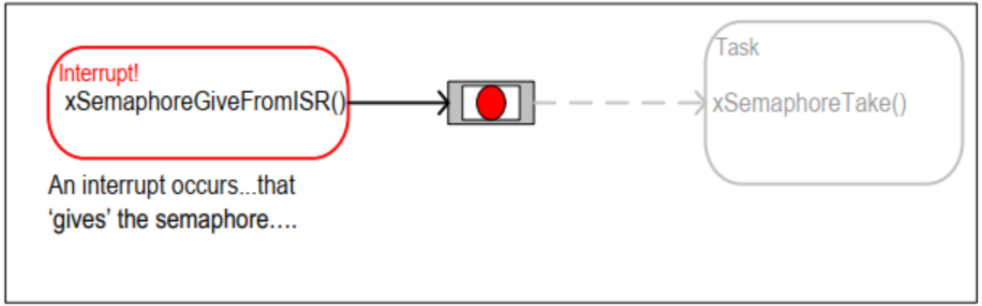
Binary semaphores

- Can be used for synchronization
- Do not support priority inheritance protocol
- Binary semaphores can be used in ISRs



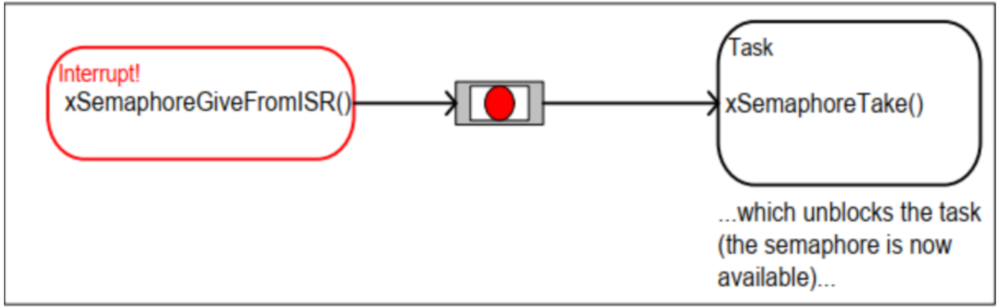
Binary semaphores

- Can be used for synchronization
- Do not support priority inheritance protocol
- Binary semaphores can be used in ISRs



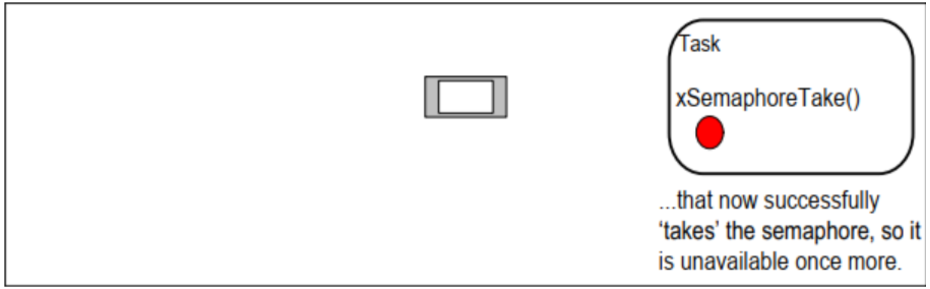
Binary semaphores

- Can be used for synchronization
- Do not support priority inheritance protocol
- Binary semaphores can be used in ISRs



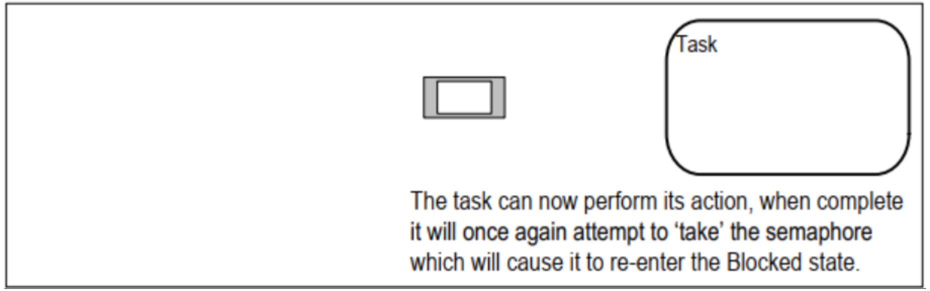
Binary semaphores

- Can be used for synchronization
- Do not support priority inheritance protocol
- Binary semaphores can be used in ISRs



Binary semaphores

- Can be used for synchronization
- Do not support priority inheritance protocol
- Binary semaphores can be used in ISRs

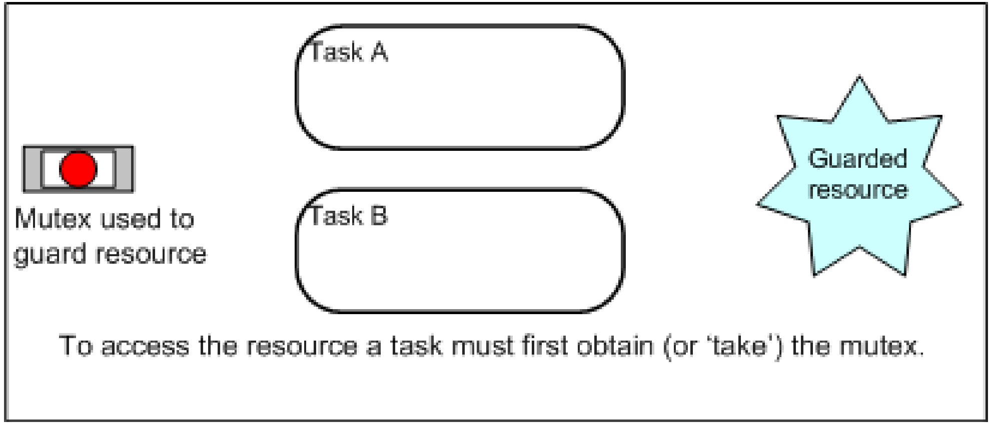


Counting Semaphores

- Counting pending events
  - An event handler (e.g. ISR) gives a semaphore S each time an event occurs (S.value++)
  - A handler task takes S each time it processes an event (S.value--)
  - When the semaphore S is created, S.value = 0
- Resource management
  - Get a resource (S.value--)
  - Release a resource (S.value++)
  - When the semaphore S is created, S.value = N (resource)

Mutexes

- Mutexes are binary semaphores that include a priority inheritance mechanism.
- A better choice for implementing simple mutual exclusion
- Mutexes should NOT be used from an interrupt
  - priority inheritance only makes sense for tasks, not ISRs
  - An ISR should not block to wait for a resource

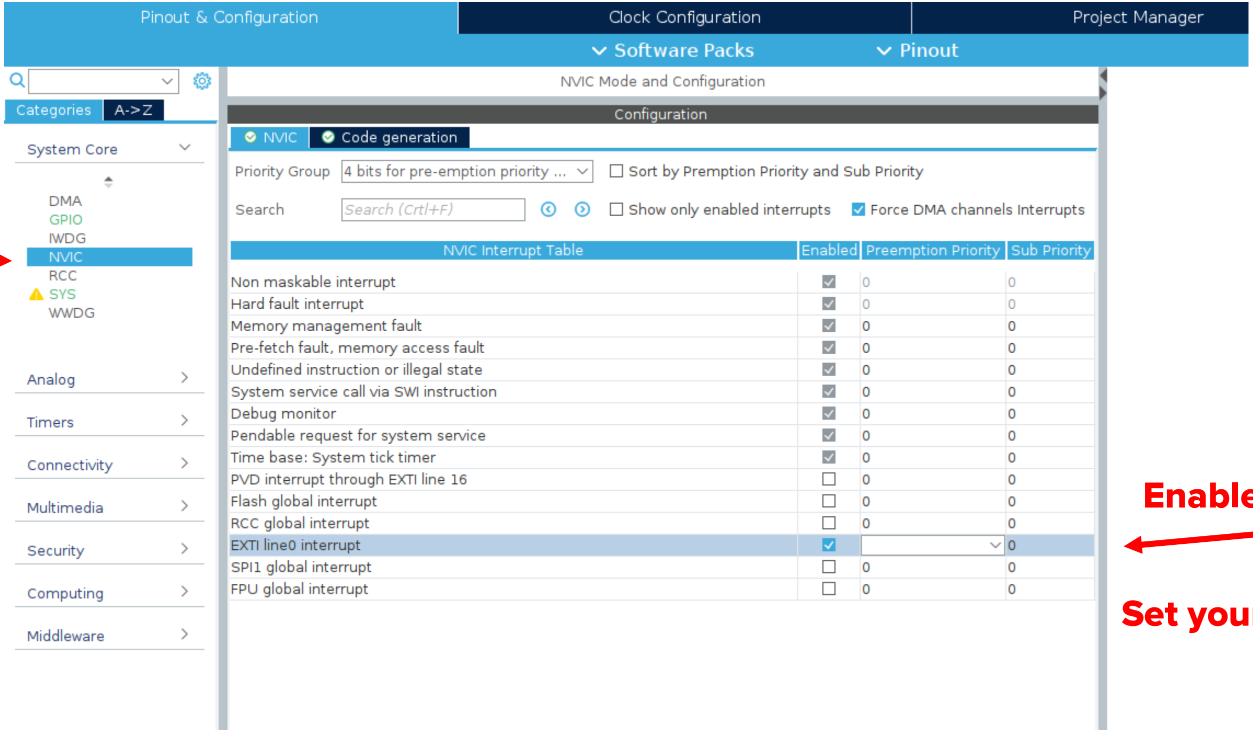


OUTLINE

Tool introduction

- Sensor interrupt
- Semaphore (include ***semphr. h***)
    - xSemaphoreCreateBinary
    - xSemaphoreGiveFromISR
    - xSemaphoreTake

設定 sensor interrupt



Enable interrupt

Set your interrupt priority

Sensor interrupt

AN3393 9.2 wake up

Register	Address	Value
CTRL_REG1	21h	01h
CTRL_REG3	23h	48h
CTRL_REG4	20h	67h
CTRL_REG5	24h	00h
THRS1_1	57h	55h
ST1_1	40h	05h
ST1_2	41h	11h
MASK1_B	59h	FCh
MASK1_A	5Ah	FCh
SETT1	5Bh	1h

ISR

- Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c

```c
/**
  * @brief  This function handles EXTI interrupt request.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}

/**
  * @brief  EXTI line detection callbacks.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

- You can define a new function” void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)”in main.c (For ISR)

```c
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

}
/* USER CODE END 4 */
```

xSemaphoreCreateBinary

```
SemaphoreHandle_t xSemaphoreCreateBinary( void );
```

Creates a binary semaphore, and returns a handle by which the semaphore can be referenced. configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h, or left undefined (in which case it will default to 1), for this RTOS API function to be available.

Return values:

NULL	The semaphore could not be created because there was insufficient FreeRTOS heap available.
Any other value	The semaphore was created successfully. The returned value is a handle by which the semaphore can be referenced.

xSemaphoreCreateBinary

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
    /* Attempt to create a semaphore. */
    xSemaphore = xSemaphoreCreateBinary();

    if( xSemaphore == NULL )
    {
        /* There was insufficient FreeRTOS heap available for the semaphore to
        be created successfully. */
    }
    else
    {
        /* The semaphore can now be used. Its handle is stored in the
        xSemahore variable.  Calling xSemaphoreTake() on the semaphore here
        will fail until the semaphore has first been given. */
    }
}
```

xSemaphoreGiveFromISR

```
xSemaphoreGiveFromISR
      (
        SemaphoreHandle_t xSemaphore,
        signed BaseType_t *pxHigherPriorityTaskWoken
      )
```

Macro to release a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR.

Parameters:

xSemaphore	A handle to the semaphore being released. This is the handle returned when the semaphore was created.
pxHigherPriorityTaskWoken	xSemaphoreGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if giving the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. From FreeRTOS V7.3.0 pxHigherPriorityTaskWoken is an optional parameter and can be set to NULL.

Returns:

pdTRUE if the semaphore was successfully given, otherwise errQUEUE_FULL.

xSemaphoreGiveFromISR

```
static uint32_t ulExampleInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken;

    /* The xHigherPriorityTaskWoken parameter must be initialized to pdFALSE as
    it will get set to pdTRUE inside the interrupt safe API function if a
    context switch is required. */
    xHigherPriorityTaskWoken = pdFALSE;

    /* 'Give' the semaphore to unblock the task, passing in the address of
    xHigherPriorityTaskWoken as the interrupt safe API function's
    pxHigherPriorityTaskWoken parameter. */
    xSemaphoreGiveFromISR( xBinarySemaphore, &xHigherPriorityTaskWoken );

    /* Pass the xHigherPriorityTaskWoken value into portYIELD_FROM_ISR().  If
    xHigherPriorityTaskWoken was set to pdTRUE inside xSemaphoreGiveFromISR()
    then calling portYIELD_FROM_ISR() will request a context switch.  If
    xHigherPriorityTaskWoken is still pdFALSE then calling
    portYIELD_FROM_ISR() will have no effect. Unlike most FreeRTOS ports, the
    Windows port requires the ISR to return a value - the return statement
    is inside the Windows version of portYIELD_FROM_ISR(). */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

xSemaphoreTake

```
xSemaphoreTake( SemaphoreHandle_t xSemaphore,
                TickType_t xTicksToWait );
```

Macro to obtain a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().

This macro must not be called from an ISR. xQueueReceiveFromISR() can be used to take a semaphore from within an interrupt if required, although this would not be a normal operation. Semaphores use queues as their underlying mechanism, so functions are to some extent interoperable.

Parameters:

xSemaphore	A handle to the semaphore being taken - obtained when the semaphore was created.
xTicksToWait	The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

Returns:

pdTRUE if the semaphore was obtained. pdFALSE if xTicksToWait expired without the semaphore becoming available.

xSemaphoreTake

```
static void vHandlerTask( void *pvParameters )
{
    /* As per most tasks, this task is implemented within an infinite loop. */
    for( ;; )
    {
        /* Use the semaphore to wait for the event.  The semaphore was created
        before the scheduler was started, so before this task ran for the first
        time.  The task blocks indefinitely, meaning this function call will only
        return once the semaphore has been successfully obtained - so there is
        no need to check the value returned by xSemaphoreTake(). */
        xSemaphoreTake( xBinarySemaphore, portMAX_DELAY );

        /* To get here the event must have occurred.  Process the event (in this
        Case, just print out a message). */
        vPrintString( "Handler task - Processing event.\r\n" );
    }
}
```

Hint

NVIC features

The nested vector interrupt controller NVIC includes the following features:

- 82 maskable interrupt channels for STM32F405xx/07xx and STM32F415xx/17xx, and up to 91 maskable interrupt channels for STM32F42xxx and STM32F43xxx (not including the 16 interrupt lines of Cortex®-M4 with FPU)
- 16 programmable priority levels (4 bits of interrupt priority are used)

but

- Handler task must have a higher priority.
- **The priority of the interrupt can be set to a value equal to configMAX_SYSCALL_INTERRUPT_PRIORITY (第23頁)**

  configMAX_SYSCALL_INTERRUPT_PRIORITY defined in FreeRTOSConfig.h

- The sensor interrupt will only be executed once if you do not read “OUTS1 (5Fh)” register in interrupt handler.

Reading this register affects the interrupt release function.

After reading OUTS1, the value is set to default (00h).

reset interrupt register

Hint

- In **Handler Task**, you must use this method instead of using "**vTaskDelay()**":
- For example for blinking LED:

```
for(int i=0;i<10;i++)
{
    uint32_t From_begin_time = HAL_GetTick();
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
    while(HAL_GetTick() - From_begin_time < 250/portTICK_RATE_MS)
    {
        ;
    }
}
```

- Template code !

Sensor interrupt

AN3393 9.2 wake up

Register	Address	Value
CTRL_REG1	21h	01h
CTRL_REG3	23h	48h
CTRL_REG4	20h	67h
CTRL_REG5	24h	00h
THRS1_1	57h	55h
ST1_1	40h	05h
ST1_2	41h	11h
MASK1_B	59h	FCh
MASK1_A	5Ah	FCh
SETT1	5Bh	1h

Motion sensor

- SM1_EN : 1

7.21 CTRL_REG1 (21h)

SM1 control register.

Table 56. SM1 control register

HYST2_1	HYST1_1	HYST0_1	-	SM1_PIN	-	-	SM1_EN

Table 57. SM1 control register structure

HYST2_1 HYST1_1 HYST0_1	Hysteresis unsigned value to be added or subtracted from threshold value in SM1 Default value: 000
SM1_PIN	0: SM1 interrupt routed to INT1; 1: SM1 interrupt routed to INT2 pin Default value: 0
SM1_EN	0: SM1 disabled; 1: SM1 enabled Default value: 0

Motion sensor

- IEA：1
- INT1_EN：1

7.23 CTRL_REG3 (23h)

Control register 3.

Table 60. Control register 3

DR_EN	IEA	IEL	INT2_EN	INT1_EN	VFILT	-	STRT

Table 61. CTRL_REG3 register description

DR_EN	DRDY signal enable to INT1. Default value: 0 (0: data ready signal not connected; 1: data ready signal connected to INT1)
IEA	Interrupt signal polarity. Default value: 0 (0: interrupt signals active LOW; 1: interrupt signals active HIGH)
IEL	Interrupt signal latching. Default value: 0 (0: interrupt signal latched; 1: interrupt signal pulsed)
INT2_EN	Interrupt 2 enable/disable. Default value:0 (0: INT2 signal disabled; 1: INT2 signal enabled)
INT1_EN	Interrupt 2 enable/disable. Default value: 0 (0: INT1/DRDY signal disabled; 1: INT1/DRDY signal enabled)
VFILT	Vector filter enable/disable. Default value: 0 (0: vector filter disabled; 1: vector filter enabled)
STRT	Soft reset bit (0: no soft reset; 1: soft reset (POR function))

Motion sensor

- ODR：0110
  - 100 Hz
- Zen：1
- Yen：1
- Xen：1

7.20 CTRL_REG4 (20h)

Control register 4.

Table 53. Control register 4

ODR3	ODR2	ODR1	ODR0	BDU	Zen	Yen	Xen

Table 54. CTRL_REG4 register description

ODR 3:0	Output data rate and power mode selection. Default value: 0000 (see *Table 55*)
BDU	Block data update. Default value: 0 (0: continuous update; 1: output registers not updated until MSB and LSB have been read)
Zen	Z-axis enable. Default value: 1 (0: Z-axis disabled; 1: Z-axis enabled)
Yen	Y-axis enable. Default value: 1 (0: Y-axis disabled; 1: Y-axis enabled)
Xen	X-axis enable. Default value: 1 (0: X-axis disabled; 1: X-axis enabled)

Motion sensor

7.38 THRS1_1 (57h)

Threshold value for SM1 operation.

Table 100. THRS1_1 register

THS7	THS6	THS5	THS4	THS3	THS2	THS1	THS0

Table 101. THRS1_1 register description

THS[7:0]	Threshold values for SM1. Default value: 0000 0000

Motion sensor

- P_X、N_X、P_Y、N_Y、P_Z、N_Z：1

7.39 **MASK1_B (59h)**

Axis and sign mask (swap) for SM1 motion-detection operation.

Table 102. MASK1_B axis and sign mask register

P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V

Table 103. MASK1_B register structure

P_X	0: X + disabled; 1: X + enabled
N_X	0: X - disabled; 1: X – enabled
P_Y	0: Y+ disabled; 1: Y + enabled
N_Y	0: Y- disabled; 1: Y – enabled
P_Z	0: Z + disabled; 1: Z + enabled
N_Z	0: Z - disabled; 1: Z – enabled
P_V	0: V + disabled; 1: V + enabled
N_V	0: V - disabled; 1: V – enabled

Motion sensor

- P_X、N_X、P_Y、N_Y、P_Z、N_Z：1

7.40 MASK1_A (5Ah)

Axis and sign mask (default) for SM1 motion-detection operation.

Table 104. MASK1_A axis and sign mask register

P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V

Table 105. MASK1_A register structure

P_X	0: X + disabled; 1: X + enabled
N_X	0: X - disabled; 1: X – enabled
P_Y	0: Y + disabled; 1: Y + enabled
N_Y	0: Y - disabled; 1: Y – enabled
P_Z	0: Z + disabled; 1: Z + enabled
N_Z	0: Z - disabled; 1: Z – enabled
P_V	0: V + disabled; 1: V + enabled
N_V	0: V - disabled; 1: V – enabled

Motion sensor

- SITR : 1

7.41 SETT1 (5Bh)

Setting of threshold, peak detection and flags for SM1 motion-detection operation.

Table 106. SETT1 register structure

P_DET	THR3_SA	ABS	-	-	THR3_MA	R_TAM	SITR

Table 107. SETT1 register description

P_DET	SM1 peak detection. Default value: 0 (0: peak detection disabled; 1: peak detection enabled)
THR3_SA	Default value: 0 (0: no action; 1: threshold 3 limit value for axis and sign mask reset (MASKB_1)
ABS	Default value: 0 (0: unsigned thresholds; 1: signed thresholds)
THR3_MA	Default value: 0 (0: no action; 1: threshold 3 limit value for axis and sign mask reset (MASKA_1)
R_TAM	Next condition validation flag. Default value: 0 (0: no valid next condition found; 1: valid next condition found and reset)
SITR	Default value: 0 (0: no actions; 1: program flow can be modified by STOP and CONT commands)

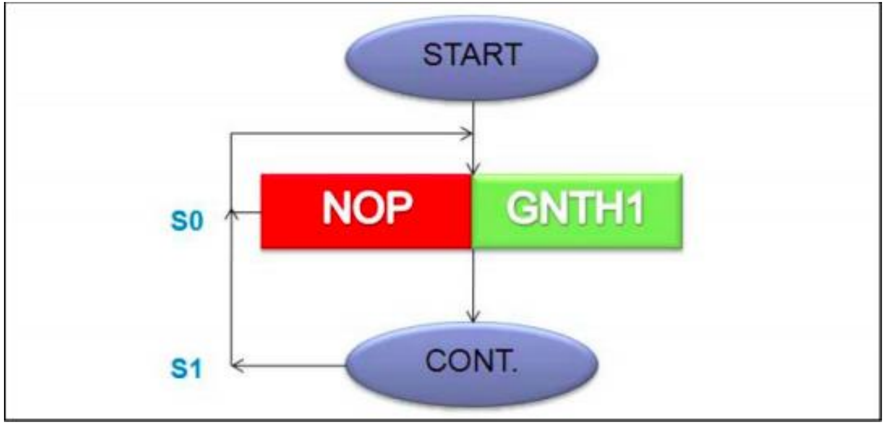
Motion sensor

7.32 STx_1 (40h-4Fh)

State machine 1 code register STx_1 (x = 1-16).

State machine 1 system register is made up of 16, 8-bit registers to implement 16-step op-code.

ST1_1	40h	05h



Motion sensor

6.1.6 GNTH1 (5h)

The GNTH1 condition is valid if any/triggered axis of the data sample set (X, Y, Z, V) is greater than threshold 1 level.

Threshold is: THRS1_y + Hysteresis.

Hysteresis is:

- State Machine 1: CTRL_REG1, bits HYST2_1, HYST1_1 and HYST0_1;
- State Machine 2: CTRL_REG2, bits HYST2_2, HYST1_2 and HYST0_2.

This condition affects or is affected by the following registers:

- THRS1_y: Threshold 1 value;
- MASKy_A and MASKy_B: Axis mask filter values;
- SETTy, bit ABS: Unsigned/signed settings;
- SETTy, bit R_TAM: Release temporary output mask settings;
- SETTy, bit P_DET: Peak detection settings;
- PEAKy: Peak output value;
- PRy: Program and Reset pointer addresses.

Motion sensor

6.2.2 CONT (11h)

The CONT command loops execution to the beginning.

This command has no parameters and it is an “Immediately executed” type.

Actions:

1. If SETTy, bit SITR = 1:
   - OUTSy is updated to selected temporary mask value;
   - Set output register (and signal if selected): STAT, bit INT_SMy = 1.
2. Default initial start executed
3. Continue execution from step address PPy = 0

This command affects or is affected by the following registers:

- State Machine y is enabled/disabled: CTRL_REG1, bit SM1_EN is set to 0/1 for State Machine 1. CTRL_REG2, bit SM2_EN is set to 0/1 for State Machine 2;
- SETTy, bit SITR: Defines output functionality of STOP command;
- OUTSy: Output value of State Machine y;
- STAT, bit INT_SMy: Indicator of valid interrupt action;
- PRy: Program and Reset pointer addresses.

Motion sensor

OUTS1 (5Fh)

Output flags on axis for interrupt SM1 management.

Table 114. OUTS1 register

P_X	N_X	P_Y	N_Y	P_Z	N_Z	P_V	N_V

Read action of this register, depending on the flag affects SM1 interrupt functions.

Table 115. OUTS1 register description

P_X	0: X + no show; 1: X+ show
N_X	0: X - no show; 1: X – show
P_Y	0: Y + no show; 1: Y + show
N_Y	0: Y - no show; 1: Y – show
P_Z	0: Z + no show; 1: Z + show
N_Z	0: Z - no show; 1: Z – show
P_V	0: V + no show; 1: V + show
N_V	0: V - no show, 1: V – show