MASTER 1 SME

RAPPORT BE

X-NUCLEO-IKS01A2

Réalisé par :
BOUDOUNET Cécile
DOUKI Thiziri

Encadré par :
Mr. PERISSE Thierry

Année universitaire 2020-2021

Table des matières

Introduction

Le **X-NUCLEO-IKS01A2** est une carte d'extension **MEMS** de mouvement et de capteur environnemental pour le STM32 Nucleo. Il est équipé de la disposition des connecteurs Arduino UNO R3 et est conçu autour de l'accéléromètre 3D et du gyroscope 3D LSM6DSL, de l'accéléromètre 3D et du magnétomètre 3D LSM303AGR, du capteur d'humidité et de température HTS221 et du capteur de pression LPS22HB. Le X-NUCLEO-IKS01A2 s'interface avec le microcontrôleur STM32 via la broche I^2C, et il est possible de changer le port I^2C par défaut.

1

1.1 Les caractéristiques du X-NUCLEO-IKS01A2 :

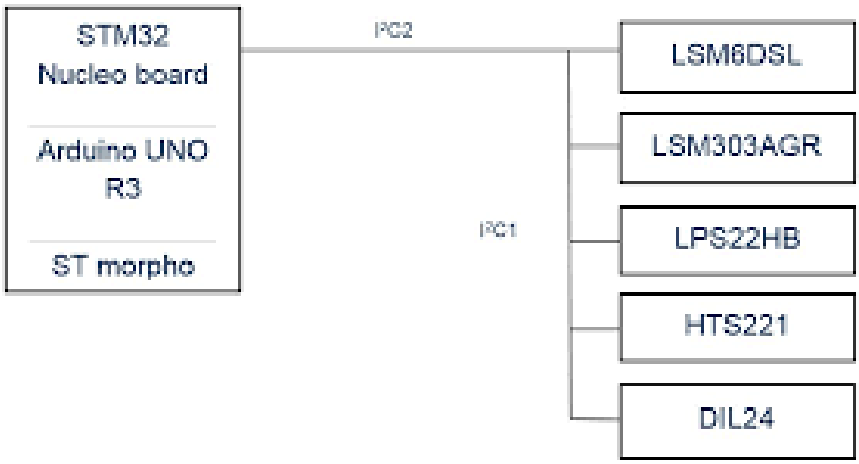
Le X-NUCLEO-IKS01A2 est composé de plusieurs capteurs :

FIGURE 1.1 – Trame de données

— Un accéléromètre 3D MEMS LSM6DSL (±2/±4/±8/±16 g) et un gyroscope 3D (±125/±245/±500/±1000/±2000 dps).
— Un accéléromètre 3D MEMS LSM303AGR (±2/±4/±8/±16 g) et magnétomètre 3D MEMS (±50 gauss).
— Un capteur de pression LPS22HB MEMS, baromètre de sortie numérique absolu

260-1260 hPa.

— Un HTS221 : humidité relative numérique capacitive et température.
— Prise DIL24 pourles adaptateurs MEMS supplémentaires et d'autres capteurs.
— Bibliothèque de micrologiciels de développement complète gratuite et des exemples pour tous les capteurs compatibles avec le micrologiciel STM32Cube.
— Compatible avec les cartes Nucleo STM32.
— Equipé d'un connecteur Arduino UNO R3.



1.2 Le protocole I2C :

Le bus I2C (Inter Integrated Circuit) a été développé pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne.

C'est un protocole de communication série, les données sont donc transférées bit par bit le long d'un seul fil.

1.2.1 Caractèristiques :

Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grace à seulement trois fils :

— Un signal de donnée (SDA).
— Un signal d'horloge (SCL).
— Un signal de référence électrique (Masse).

1.2.2 Fonctionnement :

Avec I2C, les données sont transférées dans des messages. Les messages sont divisés en trames de données. Chaque message a une trame d'adresse qui contient l'adresse binaire de l'esclave, et une ou plusieurs trames de données qui contiennent les données en cours de transmission. Le message comprend également des conditions de démarrage et d'arrêt, des bits de lecture / écriture et des bits ACK / NACK entre chaque trame de données.

- **Condition de démarrage :** La ligne SDA passe d'un niveau de tension élevé à un niveau de tension bas avant que la ligne SCL ne passe du niveau haut au niveau bas.
- **Condition d'arrêt :** La ligne SDA passe d'un niveau de tension bas à un niveau de tension élevé après que la ligne SCL passe du niveau bas au niveau haut.
- **Trame d'adresse :** Une séquence de 7 ou 10 bits unique à chaque esclave qui identifie l'esclave lorsque le maître veut communiquer avec lui.
- **Bit lecture / écriture :** Un seul bit spécifiant si le maître envoie des données à l'esclave (niveau de tension bas) ou lui demande des données (niveau de tension haut)
- **Bit ACK/NACK :** Chaque trame d'un message est suivie d'un bit d'acquittement/non-acquittement. Si une trame d'adresse ou une trame de données a été reçue avec succès, un bit ACK est renvoyé à l'expéditeur depuis le récepteur.

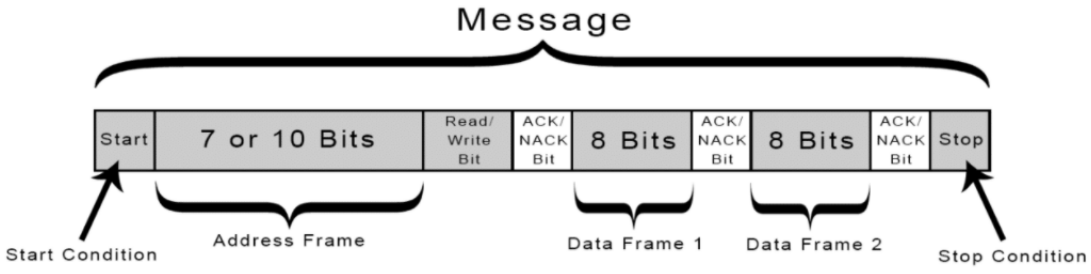


FIGURE 1.2 – Trame de données

2

La mise en oeuvre

2.1 Configurations STM32CUBEMX et utilisation de la bibliothèque MEMS :

Lors de l'utilisation des exemples d'applications, tous les composants du progiciel d'extension, y compris les applications, doivent être correctement configurés. Le progiciel **X-CUBE-MEMS1** doit être ajouté au projet et on choisit l'extension de la carte ainsi que l'application qu'on veut utiliser. Donc, dans notre cas, on selectionne l'extension **IKS01A2** et **IKS01A2_DataLogTerminal** comme application.

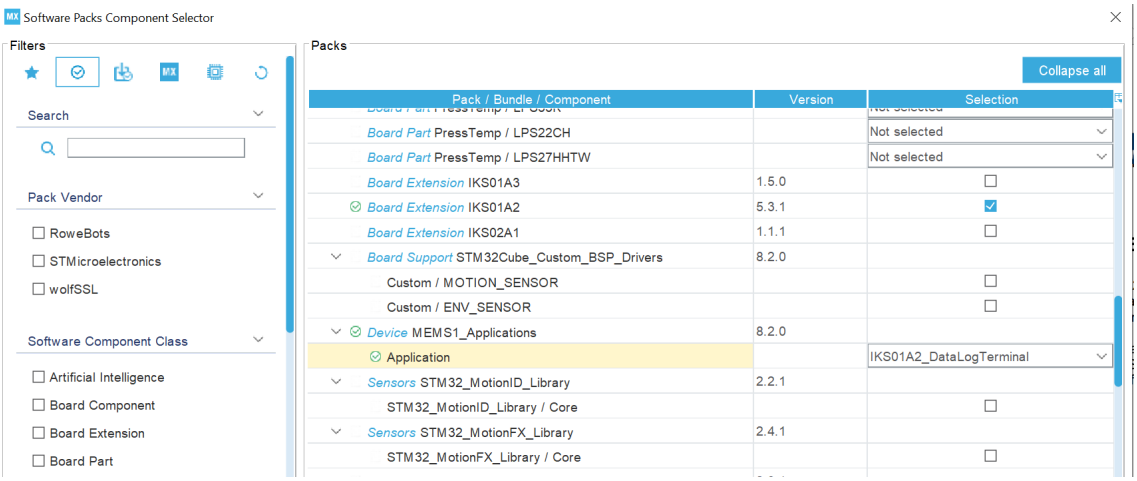


FIGURE 2.1 – Fenêtre de sélection des composants des packs logiciels STM32CubeMX

Puis on passe à la configuration des Pins celon le tableau suivant :

PIN	Mode	Etiquette
PB8	I2C1_SCL	
PB9	I2C1_SDA	
PB5	GPIO_EXTI5	
PB4	GPIO_EXTI4	
PB10	GPIO_EXTI10	
PA2	USART2_TX	USART_TX
PA3	USART2_RX	USART_RX
PA5	GPIO_Output	LD2 [Green Led]
PC13	GPIO_EXTI13	B1 [Blue PushButton]

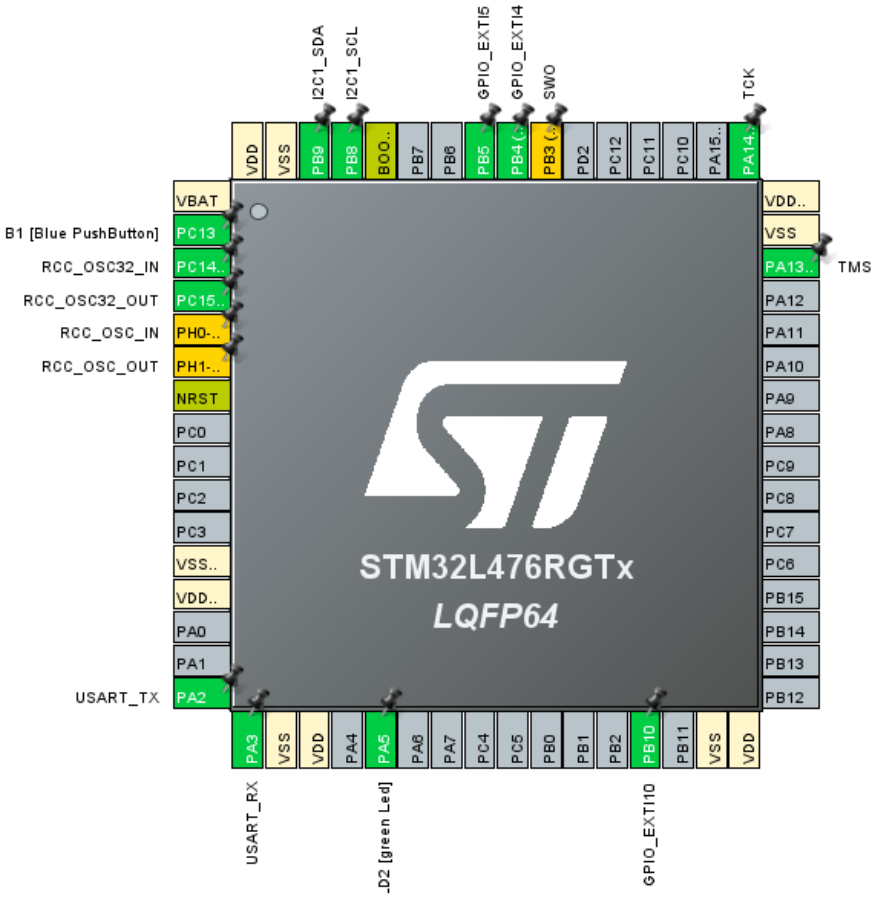


FIGURE 2.2 – Pins et configuration du STM32CubeMX

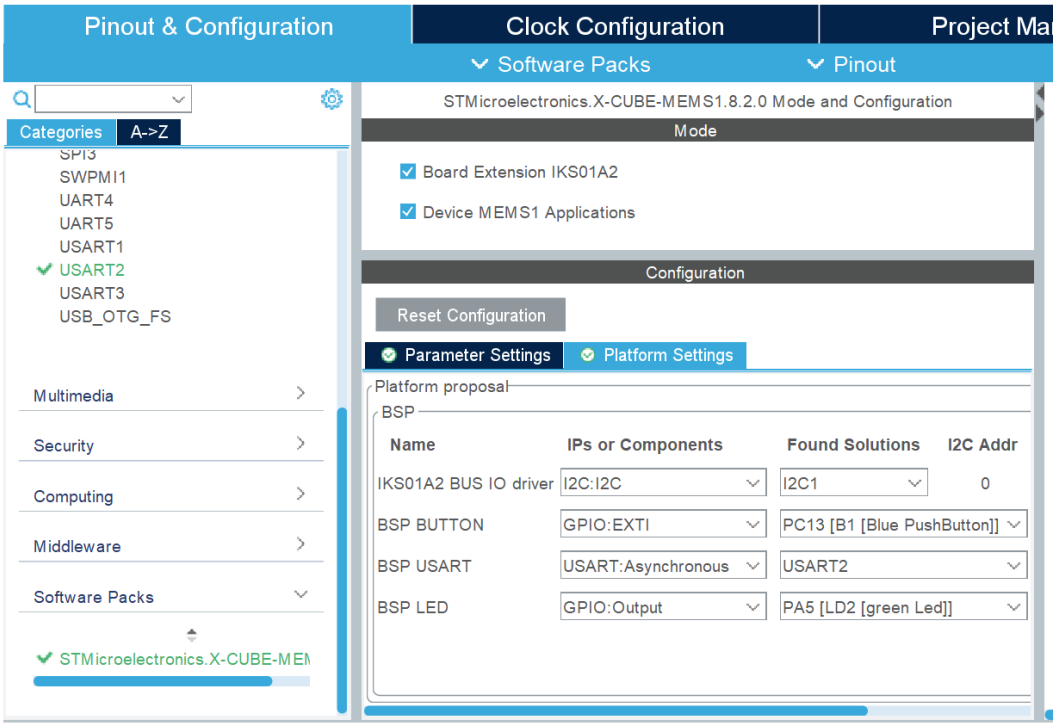


FIGURE 2.3 – Mode et configuration du STM32CubeMX

Puis on passe à la confiduration des interruptions, on active les lignes d'interruptions EXTI en choisissant NVIC :

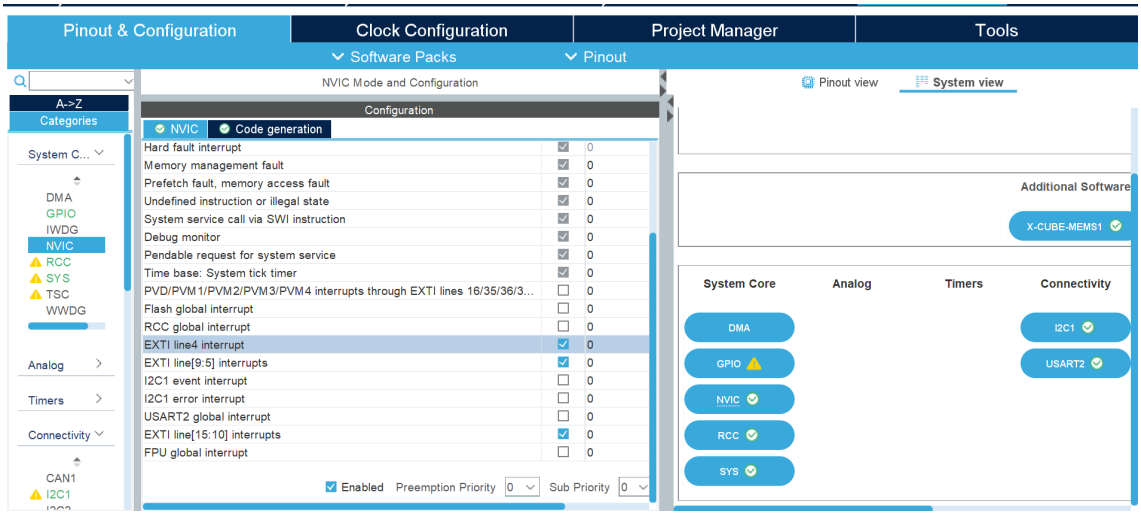


FIGURE 2.4 – Configuration des interruptions

2.2 Réalisation :

Une fois la configuration finie, on génère le projet dans lequel on trouve toutes les bibliothèques dont on aura besoin ainsi que le main. Puis on passe au cablage des composants suivant le schéma fonctionnel suivant :

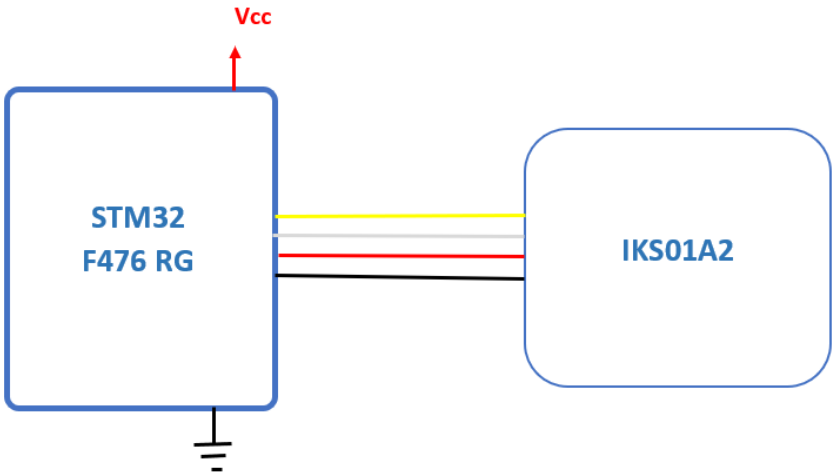


FIGURE 2.5 – Schéma fonctionnel

Par le suite on modifie le code dans le but d'utiliser que le **LSM6DSL** qui est l'accéléromètre 3D gyroscope et le magnétomètre 3D du **LSM303AGR**.

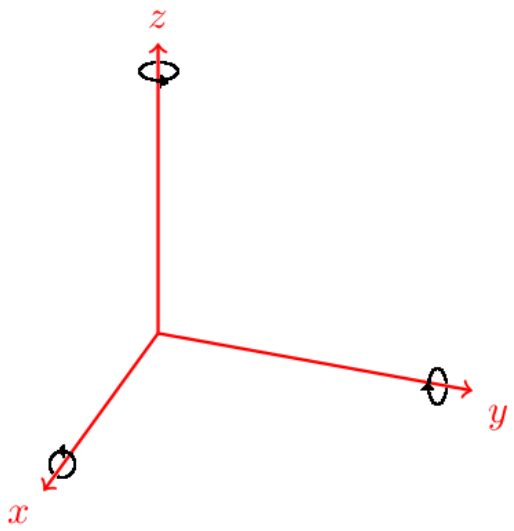
On utilise fréquemment g (accélération de la pesanteur) comme unité d'accélération. L'équation générale de la dynamique étant :

$$\sum \vec{F} = m\vec{a}$$

Tout comme la vitesse et la position, l'accélération peut être associée à un vecteur ayant même sens, donc elle peut être exprimée en fonction des ses composantes suivant les différents axes du repère 3D :

$$\vec{a} = a_x\vec{i} + a_y\vec{j} + a_z\vec{k}$$

On obtient les valeurs de l'accélération, gyroscope et champs magnétique sur les trois axes **x**, **y** et **z** ainsi que l'adresse I2C du capteur.



Quand le capteur est posé à plat et tourne sur lui même (autour de l'axe des z), la valeur de l'accélération sur l'axe des **z** reste fixe par contre les valeurs sur les axes **x** et **y** changent.

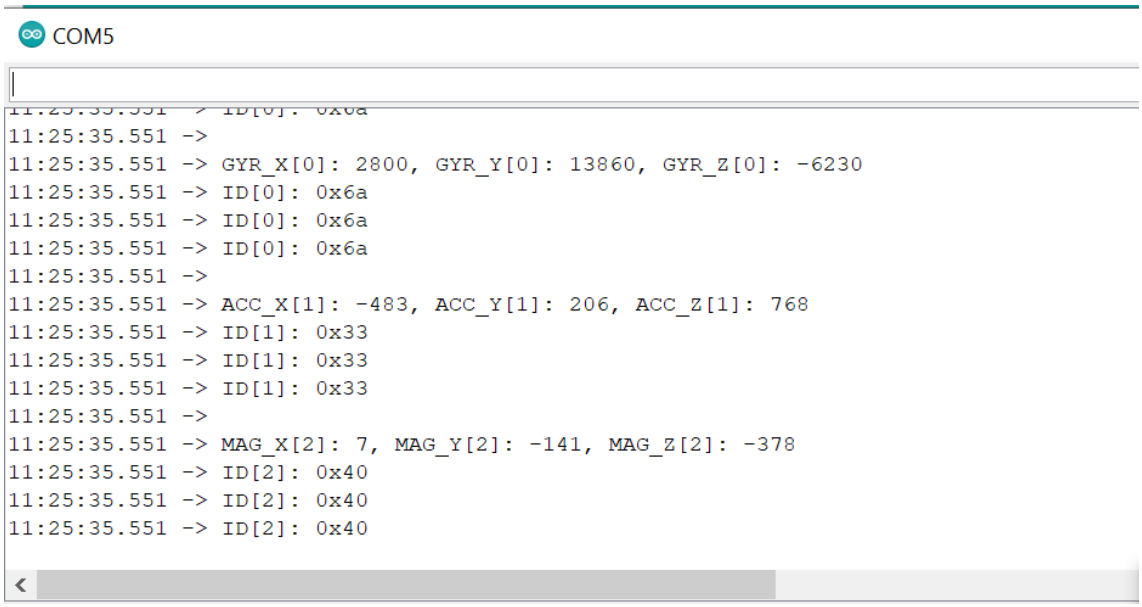


FIGURE 2.6 – Output sur le moniteur série

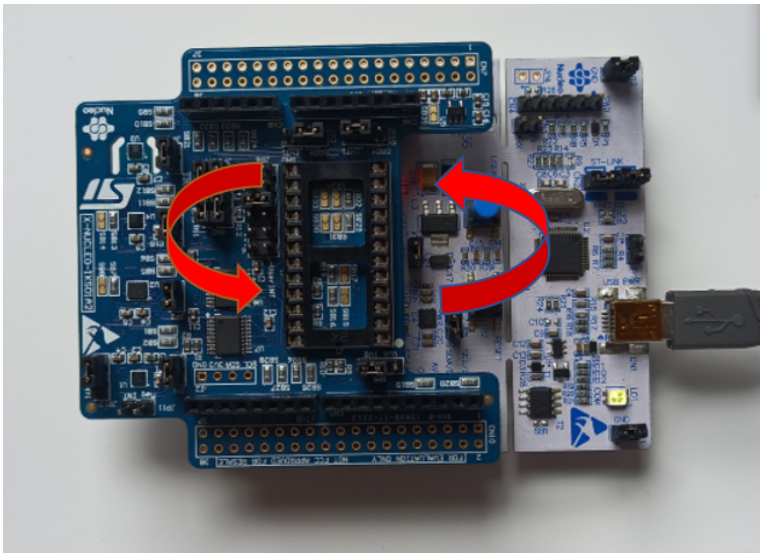


FIGURE 2.7 – Rotation autour de l'axe des z .

Quand le capteur tourne autour de l'axe des x, la valeur de l'accélération sur l'axe des **x** reste fixe par contre les valeurs sur les axes **z** et **y** changent. Quand le capteur tourne autour de l'axe des y, la valeur de l'accélération sur l'axe des **y** reste fixe par contre les valeurs sur les axes **z** et **x** changent.

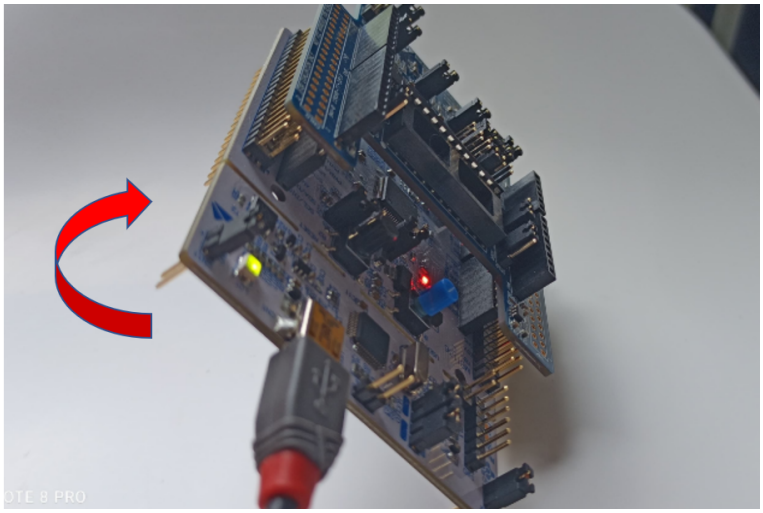


FIGURE 2.8 – Rotation autour de l'axe des x .

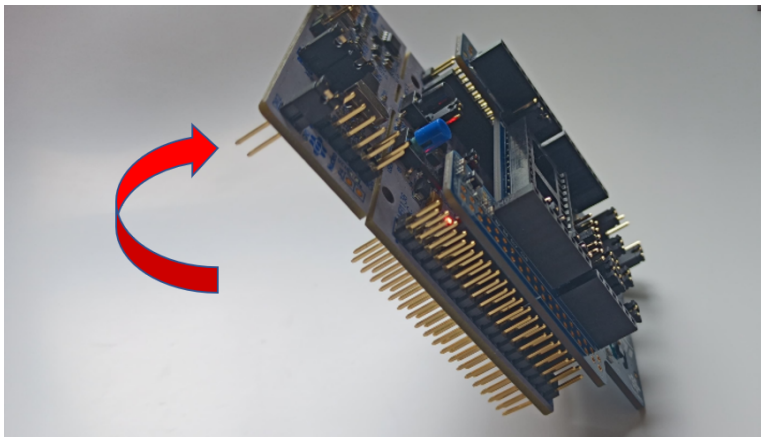


FIGURE 2.9 – Rotation autour de l'axe des y .

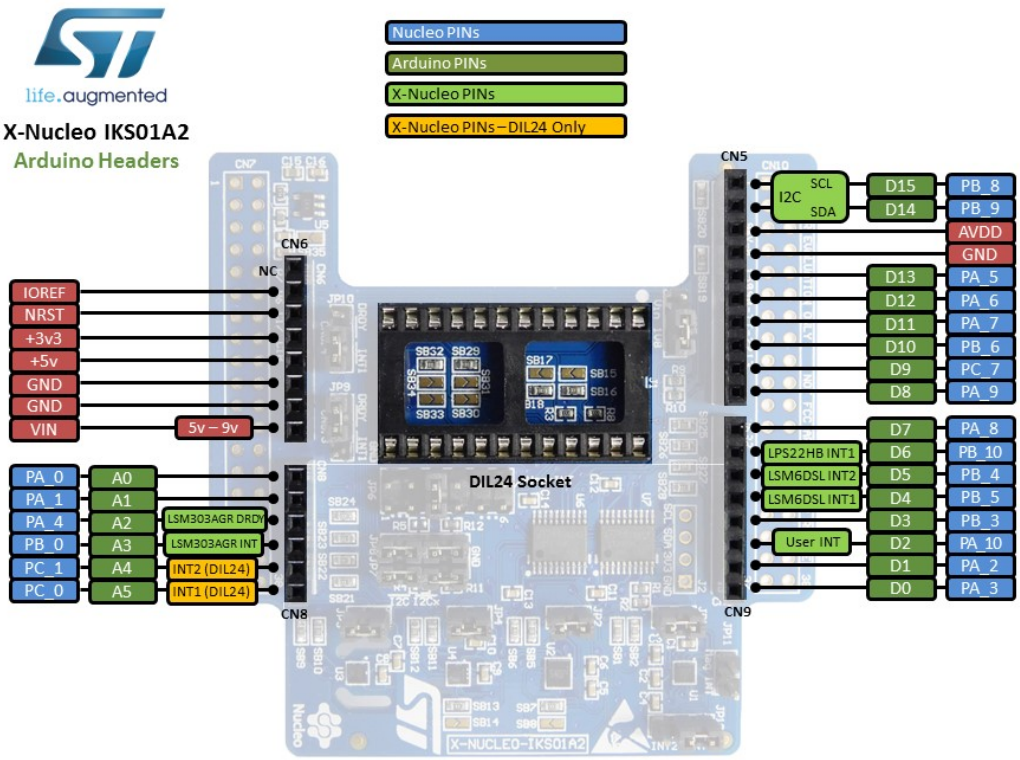
Conclusion

Ce TP Bureau d'Etude nous a permis on de manipuler le module X-NUCLEO-IKS01A2 qui contient plusieurs capteurs. Nous avons appris l'utilisations des bibliotéques et extentions de MEMS et à générer le code spécifique à chaque capteur celon notre besoin.

A

Annexe

A.1 x-nucleo-iks01a2 pinout



A.2 main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************

  ******************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "app_mems.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
```

```
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
```

```
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_MEMS_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

  MX_MEMS_Process();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators
  * in the RCC_OscInitTypeDef structure.
```

```
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
 RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4)
  != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection =
 RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_I2C1;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1)
  != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pins : PB10 PB4 PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI4_IRQn);

  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
```

```
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */

  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
```

```
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/****** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

A.3 app_mems.c

```
/**
  ******************************************
  * File Name  : app_mems.c
  * Description  : This file provides code for the configuration
  *        of the STMicroelectronics.X-CUBE-MEMS1.8.2.0 instances.
  ************************

  ****************************
  */

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "app_mems.h"
#include "main.h"
#include <stdio.h>

#include "iks01a2_motion_sensors.h"
#include "iks01a2_env_sensors.h"
#include "stm32l4xx_nucleo.h"
#include "math.h"

/* Private typedef -----------------------------------------------------------*/
typedef struct displayFloatToInt_s {
  int8_t sign; /* 0 means positive,1 means negative*/
```

```
  uint32_t  out_int;
  uint32_t  out_dec;
} displayFloatToInt_t;

/* Private define ------------------------------------------------------------*/
#define MAX_BUF_SIZE 256
------------------------------------------------------------*/
static volatile uint8_t PushButtonDetected = 0;
static uint8_t verbose = 1;
static IKS01A2_MOTION_SENSOR_Capabilities_t MotionCapabilities
[IKS01A2_MOTION_INSTANCES_NBR];
static IKS01A2_ENV_SENSOR_Capabilities_t EnvCapabilities
[IKS01A2_ENV_INSTANCES_NBR];
static char dataOut[MAX_BUF_SIZE];
static int32_t PushButtonState = GPIO_PIN_RESET;

/* Private function prototypes -----------------------------------------------*/
static void floatToInt
(float in, displayFloatToInt_t *out_value, int32_t dec_prec);
static void Accelero_Sensor_Handler(uint32_t Instance);
static void Gyro_Sensor_Handler(uint32_t Instance);
static void Magneto_Sensor_Handler(uint32_t Instance);
//static void Temp_Sensor_Handler(uint32_t Instance);
//static void Hum_Sensor_Handler(uint32_t Instance);
//static void Press_Sensor_Handler(uint32_t Instance);
static void MX_IKS01A2_DataLogTerminal_Init(void);
static void MX_IKS01A2_DataLogTerminal_Process(void);

void MX_MEMS_Init(void)
{
  /* USER CODE BEGIN SV */

  /* USER CODE END SV */

  /* USER CODE BEGIN MEMS_Init_PreTreatment */

  /* USER CODE END MEMS_Init_PreTreatment */
```

```
  /* Initialize the peripherals and the MEMS components */

  MX_IKS01A2_DataLogTerminal_Init();

  /* USER CODE BEGIN MEMS_Init_PostTreatment */

  /* USER CODE END MEMS_Init_PostTreatment */
}

/*
 * LM background task
 */
void MX_MEMS_Process(void)
{
  /* USER CODE BEGIN MEMS_Process_PreTreatment */

  /* USER CODE END MEMS_Process_PreTreatment */

  MX_IKS01A2_DataLogTerminal_Process();

  /* USER CODE BEGIN MEMS_Process_PostTreatment */

  /* USER CODE END MEMS_Process_PostTreatment */
}

/**
  * @brief  Initialize the DataLogTerminal application
  * @retval None
  */
void MX_IKS01A2_DataLogTerminal_Init(void)
{
  displayFloatToInt_t out_value_odr;
  int i;

  /* Initialize LED */
  BSP_LED_Init(LED2);
```

```
  /* Initialize button */
  BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);


  PushButtonState = (BSP_PB_GetState(BUTTON_KEY)) ? 0 : 1;

  /* Initialize Virtual COM Port */
  BSP_COM_Init(COM1);

  IKS01A2_MOTION_SENSOR_Init(IKS01A2_LSM6DSL_0, MOTION_ACCELERO
  | MOTION_GYRO);

  IKS01A2_MOTION_SENSOR_Init(IKS01A2_LSM303AGR_ACC_0,
  MOTION_ACCELERO);

  IKS01A2_MOTION_SENSOR_Init(IKS01A2_LSM303AGR_MAG_0,
  MOTION_MAGNETO);

  for(i = 0; i < IKS01A2_MOTION_INSTANCES_NBR; i++)//ACCELERO
  {
    IKS01A2_MOTION_SENSOR_GetCapabilities(i, &MotionCapabilities[i]);
    snprintf(dataOut, MAX_BUF_SIZE,
 "\r\nMotion Sensor Instance %d capabilities: \r\n ACCELEROMETER:
 %d\r\n GYROSCOPE: %d\r\n
 MAGNETOMETER: %d\r\n LOW_POWER: %d\r\n",
i, MotionCapabilities[i].Acc, MotionCapabilities[i].Gyro,
MotionCapabilities[i].Magneto, MotionCapabilities[i].LowPower);
    printf("%s", dataOut);
    floatToInt(MotionCapabilities[i].AccMaxOdr, &out_value_odr, 3);
    snprintf(dataOut, MAX_BUF_SIZE, " MAX_ACC_ODR: %d.%03d Hz,
    MAX_ACC_FS: %d\r\n",
    (int)out_value_odr.out_int,
            (int)out_value_odr.out_dec,
            (int)MotionCapabilities[i].AccMaxFS);
    printf("%s", dataOut);
    floatToInt(MotionCapabilities[i].GyroMaxOdr, &out_value_odr, 3);
```

```
    snprintf(dataOut, MAX_BUF_SIZE, " MAX GYRO ODR: %d.%03d Hz,
    MAX GYRO FS: %d\r\n",
    (int)out_value_odr.out_int,
            (int)out_value_odr.out_dec,
            (int)MotionCapabilities[i].GyroMaxFS);
    printf("%s", dataOut);
    floatToInt(MotionCapabilities[i].MagMaxOdr, &out_value_odr, 3);
    snprintf(dataOut, MAX_BUF_SIZE, " MAX MAG ODR: %d.%03d Hz,
    MAX MAG FS: %d\r\n",
    (int)out_value_odr.out_int,
            (int)out_value_odr.out_dec,
            (int)MotionCapabilities[i].MagMaxFS);
    printf("%s", dataOut);
  }


}

/**
  * @brief  BSP Push Button callback
  * @param  Button Specifies the pin connected EXTI line
  * @retval None.
  */
void BSP_PB_Callback(Button_TypeDef Button)
{
  PushButtonDetected = 1;
}

/**
  * @brief  Process of the DataLogTerminal application
  * @retval None
  */
void MX_IKS01A2_DataLogTerminal_Process(void)
{
  int i;

  if (PushButtonDetected != 0U)
```

```
  {
    /* Debouncing */
    HAL_Delay(50);

    /* Wait until the button is released */
    while ((BSP_PB_GetState( BUTTON_KEY ) == PushButtonState));

    /* Debouncing */
    HAL_Delay(50);

    /* Reset Interrupt flag */
    PushButtonDetected = 0;

    /* Do nothing */
  }

  for(i = 0; i < IKS01A2_MOTION_INSTANCES_NBR; i++)
  {
    if(MotionCapabilities[i].Acc)
    {
      Accelero_Sensor_Handler(i);
    }
    if(MotionCapabilities[i].Gyro)
    {
      Gyro_Sensor_Handler(i);
    }
    if(MotionCapabilities[i].Magneto)
    {
      Magneto_Sensor_Handler(i);
    }
  }



  HAL_Delay( 1000 );
}
```

```
/**
  * @brief  Splits a float into two integer values.
  * @param  in the float value as input
  * @param  out_value the pointer to the output integer structure
  * @param  dec_prec the decimal precision to be used
  * @retval None
  */
static void floatToInt(float in, displayFloatToInt_t *out_value,
int32_t dec_prec)
{
  if(in >= 0.0f)
  {
    out_value->sign = 0;
  }else
  {
    out_value->sign = 1;
    in = -in;
  }

  in = in + (0.5f / pow(10, dec_prec));
  out_value->out_int = (int32_t)in;
  in = in - (float)(out_value->out_int);
  out_value->out_dec = (int32_t)trunc(in * pow(10, dec_prec));
}

/**
  * @brief  Handles the accelerometer axes data getting/sending
  * @param  Instance the device instance
  * @retval None
  */
static void Accelero_Sensor_Handler(uint32_t Instance)
{
  float odr;
  int32_t fullScale;
  IKS01A2_MOTION_SENSOR_Axes_t acceleration;
  displayFloatToInt_t out_value;
  uint8_t whoami;
```

```
  if (IKS01A2_MOTION_SENSOR_GetAxes(Instance, MOTION_ACCELERO,
  &acceleration))
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nACC[%d]: Error\r\n",
    (int)Instance);
  }
  else
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nACC_X[%d]: %d, ACC_Y[%d]: %d,
    ACC_Z[%d]: %d\r\n", (int)Instance,
             (int)acceleration.x, (int)Instance, (int)acceleration.y,
             (int)Instance, (int)acceleration.z);
  }

  printf("%s", dataOut);

  if (verbose == 1)
  {
    if (IKS01A2_MOTION_SENSOR_ReadID(Instance, &whoami))
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: Error\r\n",
      (int)Instance);
    }
    else
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: 0x%x\r\n",
      (int)Instance, (int)whoami);
    }

    printf("%s", dataOut);

    if (IKS01A2_MOTION_SENSOR_GetOutputDataRate(Instance,
    MOTION_ACCELERO, &odr))
    {
//      snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
ERROR\r\n", (int)Instance);
```

```
    }
    else
    {
      floatToInt(odr, &out_value, 3);
    //  snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
     %d.%03d Hz\r\n",
    (int)Instance, (int)out_value.out_int,
            //       (int)out_value.out_dec);
    }

    printf("%s", dataOut);

    if (IKS01A2_MOTION_SENSOR_GetFullScale(Instance,
    MOTION_ACCELERO, &fullScale))
    {
    //  snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]:
     ERROR\r\n", (int)Instance);
    }
    else
    {
    //      snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]: %d g\r\n",
    (int)Instance, (int)fullScale);
    }

    printf("%s", dataOut);
  }
}

/**
  * @brief  Handles the gyroscope axes data getting/sending
  * @param  Instance the device instance
  * @retval None
  */
static void Gyro_Sensor_Handler(uint32_t Instance)
{
  float odr;
  int32_t fullScale;
```

```
  IKS01A2_MOTION_SENSOR_Axes_t angular_velocity;
  displayFloatToInt_t out_value;
  uint8_t whoami;

  if (IKS01A2_MOTION_SENSOR_GetAxes(Instance, MOTION_GYRO,
  &angular_velocity))
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nGYR[%d]: Error\r\n",
    (int)Instance);
  }
  else
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nGYR_X[%d]: %d,
      GYR_Y[%d]: %d,
      GYR_Z[%d]: %d\r\n", (int)Instance,
                (int)angular_velocity.x, (int)Instance,
                (int)angular_velocity.y,
                (int)Instance, (int)angular_velocity.z);
  }

  printf("%s", dataOut);

  if (verbose == 1)
  {
    if (IKS01A2_MOTION_SENSOR_ReadID(Instance, &whoami))
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: Error\r\n",
      (int)Instance);
    }
    else
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: 0x%x\r\n",
      (int)Instance, (int)whoami);
    }

    printf("%s", dataOut);
```

```
    if (IKS01A2_MOTION_SENSOR_GetOutputDataRate(Instance,
    MOTION_GYRO, &odr))
    {
    //    snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
     ERROR\r\n", (int)Instance);
    }
    else
    {
      floatToInt(odr, &out_value, 3);
    //    snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
      %d.%03d Hz\r\n",
    (int)Instance, (int)out_value.out_int,
          //      (int)out_value.out_dec);
    }

    printf("%s", dataOut);

    if (IKS01A2_MOTION_SENSOR_GetFullScale(Instance,
    MOTION_GYRO, &fullScale))
    {
    //    snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]: ERROR\r\n",
    (int)Instance);
    }
    else
    {
    //   snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]: %d dps\r\n",
    (int)Instance, (int)fullScale);
    }

    printf("%s", dataOut);
  }
}

/**
  * @brief  Handles the magneto axes data getting/sending
  * @param  Instance the device instance
  * @retval None
```

```
  */
static void Magneto_Sensor_Handler(uint32_t Instance)
{
  float odr;
  int32_t fullScale;
  IKS01A2_MOTION_SENSOR_Axes_t magnetic_field;
  displayFloatToInt_t out_value;
  uint8_t whoami;

  if (IKS01A2_MOTION_SENSOR_GetAxes(Instance,
  MOTION_MAGNETO, &magnetic_field))
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nMAG[%d]: Error\r\n",
    (int)Instance);
  }
  else
  {
    snprintf(dataOut, MAX_BUF_SIZE, "\r\nMAG_X[%d]: %d, MAG_Y[%d]:
      %d, MAG_Z[%d]: %d\r\n",
    (int)Instance,
    (int)magnetic_field.x, (int)Instance, (int)magnetic_field.y,
    (int)Instance, (int)magnetic_field.z);
  }

  printf("%s", dataOut);

  if (verbose == 1)
  {
    if (IKS01A2_MOTION_SENSOR_ReadID(Instance, &whoami))
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: Error\r\n",
      (int)Instance);
    }
    else
    {
      snprintf(dataOut, MAX_BUF_SIZE, "ID[%d]: 0x%x\r\n",
      (int)Instance, (int)whoami);
```

```
    }

    printf("%s", dataOut);

    if (IKS01A2_MOTION_SENSOR_GetOutputDataRate(Instance,
    MOTION_MAGNETO, &odr))
    {
    //    snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
      ERROR\r\n", (int)Instance);
    }
    else
    {
      floatToInt(odr, &out_value, 3);
    //    snprintf(dataOut, MAX_BUF_SIZE, "Output Data Rate[%d]:
      %d.%03d Hz\r\n",
    (int)Instance, (int)out_value.out_int,
                  //  (int)out_value.out_dec);
    }

    printf("%s", dataOut);

    if (IKS01A2_MOTION_SENSOR_GetFullScale(Instance,
    MOTION_MAGNETO, &fullScale))
    {
    //    snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]: ERROR\r\n",
    (int)Instance);
    }
    else
    {
    //   snprintf(dataOut, MAX_BUF_SIZE, "Full Scale[%d]:
       %d gauss\r\n",
    (int)Instance, (int)fullScale);
    }

    printf("%s", dataOut);
  }
}
```

```
/**
  * @brief  Handles the temperature data getting/sending
  * @param  Instance the device instance
  * @retval None
  */


/**
  * @brief  Handles the pressure sensor data getting/sending
  * @param  Instance the device instance
  * @retval None
  */




/**
  * @brief  Handles the humidity data getting/sending
  * @param  Instance the device instance
  * @retval None
  */



#ifdef __cplusplus
}
#endif

/** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```