```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define LIS3MDL_ADDRESS 0x1E
#define LIS3MDL_WHO_AM_I 0x0F //Default: 0011 1101b = 0x3D
#define LIS3MDL_CTRL_REG1 0x20
#define LIS3MDL_CTRL_REG2 0x21
#define LIS3MDL_CTRL_REG3 0x22
#define LIS3MDL_CTRL_REG4 0x23
#define LIS3MDL_STATUS_REG 0x27
#define LIS3MDL_OUT_X_L 0x28
#define LIS3MDL_OUT_X_H 0x29
#define LIS3MDL_OUT_Y_L 0x2A
#define LIS3MDL_OUT_Y_H 0x2B
#define LIS3MDL_OUT_Z_L 0x2C
#define LIS3MDL_OUT_Z_H 0x2D
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
```

```c
uint8_t userButtonFlag = 0;
uint8_t i2cMemRead[4] = {0, 0, 0, 0};
uint8_t i2cMemWrite[4] = {0, 0, 0, 0};

typedef struct LIS3MDL_RegisterContent{
	uint8_t whoAmI;
	uint8_t ctrlReg1;
	uint8_t ctrlReg2;
	uint8_t ctrlReg3;
	uint8_t ctrlReg4;
	uint8_t statusReg;
	uint8_t outX[2];
	uint8_t outY[2];
	uint8_t outZ[2];
}LIS3MDL_RegisterContentTypeDef;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
int __io_putchar(int ch){
	HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
	return ch;
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == USER_BUTTON_Pin) userButtonFlag = 1;
}
/* 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, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
```

```c
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  LIS3MDL_RegisterContentTypeDef magnetometr;
  //ODCZYT REJESTROW MAGNETOMETRU W CELU POZNANIA USTAWIEN
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_WHO_AM_I,
I2C_MEMADD_SIZE_8BIT, &magnetometr.whoAmI, 1, HAL_MAX_DELAY);
  printf("WHO AM I: %hx\r\n", magnetometr.whoAmI);
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG1,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg1, 1, HAL_MAX_DELAY);
  printf("CTRL REG 1: %hx\r\n", magnetometr.ctrlReg1);
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG2,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg2, 1, HAL_MAX_DELAY);
  printf("CTRL REG 2: %hx\r\n", magnetometr.ctrlReg2);
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG3,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg3, 1, HAL_MAX_DELAY);
  printf("CTRL REG 3: %hx\r\n", magnetometr.ctrlReg3);
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG4,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg4, 1, HAL_MAX_DELAY);
  printf("CTRL REG 4: %hx\r\n", magnetometr.ctrlReg4);
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_STATUS_REG,
I2C_MEMADD_SIZE_8BIT, &magnetometr.statusReg, 1, HAL_MAX_DELAY);
  printf("STATUS REG: %hx\r\n", magnetometr.statusReg);

  //PRZEPROWADZENIE SOFTWARE RESET
  magnetometr.ctrlReg2 = 0x04;
  printf("WRITE TO CTRL REG 2: %hx\r\n", magnetometr.ctrlReg2);
  HAL_I2C_Mem_Write(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG2,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg2, 1, HAL_MAX_DELAY);
  do{
        HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG2,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg2, 1, HAL_MAX_DELAY);
        printf("CTRL REG 2: %hx  TRWA SOFTWARE RESET!\r\n", magnetometr.ctrlReg2);
  }while(magnetometr.ctrlReg2 & 0x04);

  //WYSWIETLENIE REJESTROW KONFIGURACYJNYCH
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG1 | 0x08,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg1, 4, HAL_MAX_DELAY);
  printf("CTRL REG 1: %hx\r\nCTRL REG 2: %hx\r\nCTRL REG 3: %hx\r\nCTRL REG 4:
%hx\r\n", magnetometr.ctrlReg1, magnetometr.ctrlReg2, magnetometr.ctrlReg3,
magnetometr.ctrlReg4);

  //PREZPROWADZENIE REBOOTU
  magnetometr.ctrlReg2 = 0x08;
  printf("WRITE TO CTRL REG 2: %hx\r\n", magnetometr.ctrlReg2);
  HAL_I2C_Mem_Write(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG2,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg2, 1, HAL_MAX_DELAY);
  do{
        HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG2,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg2, 1, HAL_MAX_DELAY);
        printf("CTRL REG 2: %hx  TRWA REBOOT!\r\n", magnetometr.ctrlReg2);
  }while(magnetometr.ctrlReg2 & 0x08);
```

```c
  //WYSWIETLENIE REJESTROW KONFIGURACYJNYCH
  HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG1 | 0x08,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg1, 4, HAL_MAX_DELAY);
  printf("CTRL REG 1: %hx\r\nCTRL REG 2: %hx\r\nCTRL REG 3: %hx\r\nCTRL REG 4:
%hx\r\n", magnetometr.ctrlReg1, magnetometr.ctrlReg2, magnetometr.ctrlReg3,
magnetometr.ctrlReg4);

  //URUCHOMIENIE CIAGLEJ KONWERSJI
  magnetometr.ctrlReg3 = 0x00;
  printf("WRITE TO CTRL REG 3: %hx\r\n", magnetometr.ctrlReg3);
  HAL_I2C_Mem_Write(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG3,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg3, 1, HAL_MAX_DELAY);
  do{
        HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_CTRL_REG3,
I2C_MEMADD_SIZE_8BIT, &magnetometr.ctrlReg3, 1, HAL_MAX_DELAY);
        printf("CTRL REG 3: %hx  TRWA ZAPIS CTRL_REG3!\r\n",
magnetometr.ctrlReg3);
   }while(magnetometr.ctrlReg3 & 0x03);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //ODCZYTAJ POMIARY JESLI SA GOTOWE
        HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_STATUS_REG,
I2C_MEMADD_SIZE_8BIT, &magnetometr.statusReg, 1, HAL_MAX_DELAY);
        printf("STATUS REG: %hx\r\n", magnetometr.statusReg);
        if(magnetometr.statusReg & 0x8){
             HAL_I2C_Mem_Read(&hi2c1, LIS3MDL_ADDRESS<<1, LIS3MDL_OUT_X_L | 0x8,
I2C_MEMADD_SIZE_8BIT, magnetometr.outX, 6, HAL_MAX_DELAY);
             printf("OUT X L: %hx  OUT X H: %hx\r\nOUT Y L: %hx  OUT Y H:
%hx\r\nOUT Z L: %hx  OUT Z H: %hx\r\n",\
                          magnetometr.outX[0], magnetometr.outX[1],
magnetometr.outY[0], magnetometr.outY[1], magnetometr.outZ[0],
magnetometr.outZ[1]);
        }
        if(userButtonFlag == 1){
              userButtonFlag = 0;
              HAL_GPIO_TogglePin(USER_LED_GPIO_Port, USER_LED_Pin);
        }
        HAL_Delay(500);
    /* USER CODE END WHILE */

    /* 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};
```

```
  /** Initializes the RCC Oscillators according to the specified parameters
  * 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_NONE;
  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_HSI;
  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_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/* 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 */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         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 can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
```

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