Compte-rendu TEST

(Date : 12/03/24)

Etudiante : Malek SAHLI

Cursus : Master 1 LBD

Année universitaire 2023/2024

Partie A

Question 1 :

- Quelle est l'adresse I2C du capteur (7 bits) ?

  -L'adresse de I2C de IS2DW12 sensor est 0x29 (7 bits).

Question 2 :

- Quelle grandeur physique ce capteur mesure-t-il ?
  - Pression différentielle
  - Pression absolue
  -Température

Question 3 :

- Quelle est la fréquence maximale de récupération des donnees (ODR) ?
  - Le débit de données maximal (ODR) du capteur IS2DW12 est de 100 Hz.

Question 4 :

- Quelle est le format des données en Provence du capteur ?

  - Le capteur IS2DW12 génère des données au format suivant :

  • Pression : entier signé 24 bits

  • Température : entier signé 16 bits

Question 5 :

- Citer des applications possibles pour ce capteur ?

  - Voici quelques exemples d'applications possibles pour le capteur IS2DW12 :

    • Mesure de la pression barométrique

    • Contrôle CVC

    • Automatisation industrielle

Partie B :

L'analyse de l'application :

Initialisation des périphériques : l'initialisation des périphériques STM32 nécessaires, tels que GPIO, UART (USART2) et I2C (I2C1), en appelant les fonctions HAL_Init(), SystemClock_Config(), MX_GPIO_Init(), MX_USART2_UART_Init() et MX_I2C1_Init().

Initialisation du contexte de l'accéléromètre : dev_ctx est initialisé avec des fonctions de lecture , écriture et de temporisation spécifiques à la plate-forme pour l'accéléromètre LIS2DW12. Il est utilisé par les fonctions du pilote LIS2DW12 pour communiquer avec le capteur via I2C.

Vérification de l'identifiant du dispositif : La fonction lis2dw12_device_id_get() est appelée pour obtenir l'identifiant du dispositif. Si l'identifiant ne correspond pas à celui attendu (LIS2DW12_ID), le programme entre dans une boucle infinie, ce qui peut être interprété comme une gestion d'erreur en cas de non-détection du capteur.

Réinitialisation et configuration de l'accéléromètre : On réinitialise et configure ensuite l'accéléromètre LIS2DW12 avec différentes options telles que le bloc de mise à jour des données, l'échelle complète, le chemin de filtrage, le mode d'alimentation et le débit de données de sortie

```
int main(void)
{
  /* USER CODE BEGIN 1 */
      stmdev_ctx_t dev_ctx;
    dev_ctx.write_reg = platform_write;
      dev_ctx.read_reg = platform_read;
      dev_ctx.mdelay = HAL_Delay;
    dev_ctx.handle = &hi2c1;
  /* 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();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  /* Check device ID */
     lis2dw12_device_id_get(&dev_ctx, &whoamI);

     if (whoamI != LIS2DW12_ID)
       while (1) {
         /* manage here device not found */
       }

     /* Restore default configuration */
     lis2dw12_reset_set(&dev_ctx, PROPERTY_ENABLE);

     do {
       lis2dw12_reset_get(&dev_ctx, &rst);
     } while (rst);

     /* Enable Block Data Update */
     lis2dw12_block_data_update_set(&dev_ctx, PROPERTY_ENABLE);
     /* Set full scale */
     //lis2dw12_full_scale_set(&dev_ctx, LIS2DW12_8g);
```

```
    lis2dw12_full_scale_set(&dev_ctx, LIS2DW12_2g);
    /* Configure filtering chain
     * Accelerometer - filter path / bandwidth
     */
    lis2dw12_filter_path_set(&dev_ctx, LIS2DW12_LPF_ON_OUT);
    lis2dw12_filter_bandwidth_set(&dev_ctx, LIS2DW12_ODR_DIV_4);
    /* Configure power mode */
    lis2dw12_power_mode_set(&dev_ctx, LIS2DW12_HIGH_PERFORMANCE);
    //lis2dw12_power_mode_set(&dev_ctx,
LIS2DW12_CONT_LOW_PWR_LOW_NOISE_12bit);
    /* Set Output Data Rate */
    lis2dw12_data_rate_set(&dev_ctx, LIS2DW12_XL_ODR_25Hz);
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
        uint8_t reg;
              /* Read output only if new value is available */
              lis2dw12_flag_data_ready_get(&dev_ctx, &reg);

              if (reg) {
                  /* Read acceleration data */
                  memset(data_raw_acceleration, 0x00, 3 * sizeof(int16_t));
                  lis2dw12_acceleration_raw_get(&dev_ctx,
data_raw_acceleration);
                  //acceleration_mg[0] =
lis2dw12_from_fs8_lp1_to_mg(data_raw_acceleration[0]);
                  //acceleration_mg[1] =
lis2dw12_from_fs8_lp1_to_mg(data_raw_acceleration[1]);
                  //acceleration_mg[2] =
lis2dw12_from_fs8_lp1_to_mg(data_raw_acceleration[2]);
                  acceleration_mg[0] = lis2dw12_from_fs2_to_mg(

data_raw_acceleration[0]);
                  acceleration_mg[1] = lis2dw12_from_fs2_to_mg(

data_raw_acceleration[1]);
                  acceleration_mg[2] = lis2dw12_from_fs2_to_mg(

data_raw_acceleration[2]);
                  sprintf((char *)tx_buffer,
                              "Acceleration [mg]:%4.2f\t%4.2f\t%4.2f\r\n",
                              acceleration_mg[0], acceleration_mg[1],
acceleration_mg[2]);
                  //HAL_UART_Transmit(&huart2, tx_buffer, strlen((char
const *)tx_buffer), 1000);
                  printf((char *)tx_buffer);
              }

  }
  /* USER CODE END 3 */
```