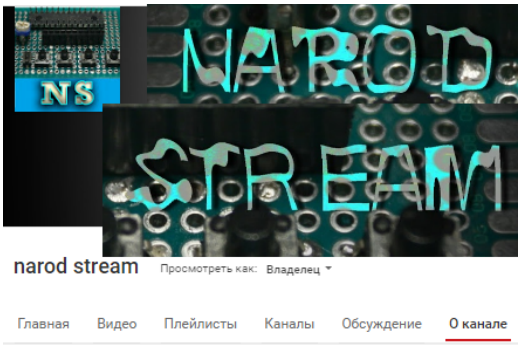
Сайт

narodstream.ru

создан в поддержку
канала YouTube

NAROD STREAM



Свежие записи

- STM32. Урок 87. LAN. ENC28J60. HTTP Server. Подключаем карту SD
- AVR. Урок 52. LAN. ENC28J60. HTTP Server. Подключаем карту SD. Часть 2
- AVR. Урок 52. LAN. ENC28J60. HTTP Server. Подключаем карту SD. Часть 1
- STM32. Урок 88. SD. SPI. FATFS. Часть 4
- STM32. Урок 88. SD. SPI. FATFS. Часть 3

Рубрики

- Программирование AVR
- Программирование STM32

STM Урок 42. Подключаем

Мета

- Регистрация
- Войти
- RSS записей

акселерометр LSM6DS0. Часть 1

Урок 42

Часть 1

Подключаем акселерометр LSM6DS0

Предыдущий урок
Программирование МК STM32
Следующая часть

Сегодня мы рассмотрим датчик, который в себе объединяет сразу два функционала – акселерометр и гироскоп. Данный акселерометр – это также акселерометр, выполненный с использованием технологии MEMS – **LSM6DS0**. Установлен он на плате расширения **X-NUCLEO-IKS01A1**, предназначенной для работы с отладочной платой Nucleo. Мы будем подключать данную оценочную плату к плате Nucleo STM32F401RE.

Данный акселерометр-гироскоп также наряду с интерфейсом I2C может подключиться и с использованием интерфейса SPI. Но мы будем использовать подключение именно по I2C, так как именно такое подключение имеет место в оценочной плате X-NUCLEO-IKS01A.

Также использовать в рамках данного занятия мы данный датчик будем только как акселерометр. В качестве гироскопа мы его подключим на более поздних занятиях.

Акселерометр в данном датчике имеет следующие технические характеристики:

Диапазон показаний ±2g/±4g/±8g/±16g;
Чувствительность 0.061 – 0.73 mg/digit;
Отклонение от нуля ±90 mg.

С некоторыми остальными показателями, регистрами, значениями и другими тонкостями акселерометра мы познакомимся в ходе его программирования.

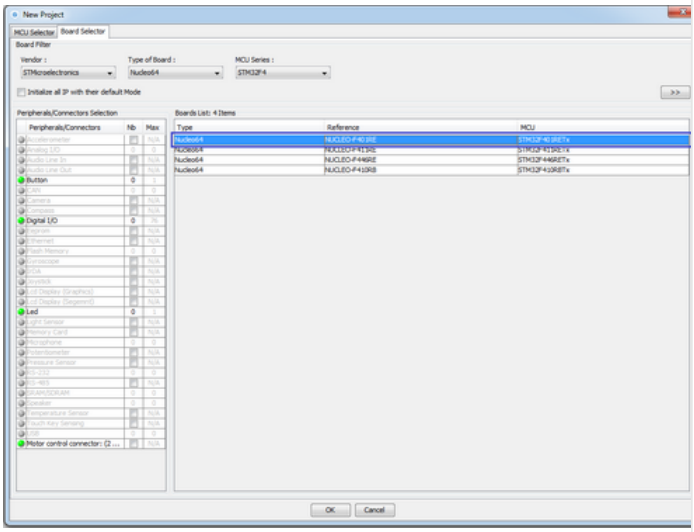
Создадим проект для в Cube MX. Выбирать мы будем не контроллер, а отладочную плату (нажмите на картинку для увеличения изображения)

- RSS комментариев
- WordPress.org

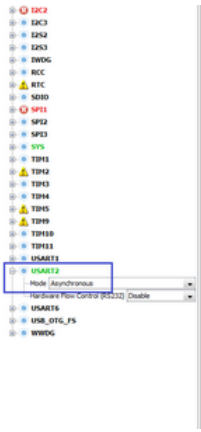
Уроки по программированию МК

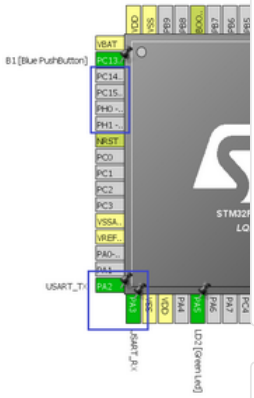
Программирование МК AVR
Программирование МК STM32

искать здесь …
Фильтровать

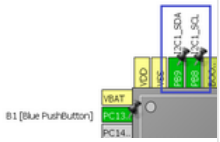


Отключим ножки портов PH0, PH1, PC14, PC15. USART 2 Включим в режим «Asynchronous» (нажмите на картинку для увеличения изображения)

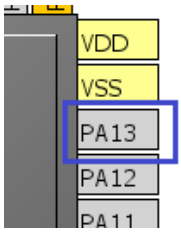




Включим I2C1, только лапки SCL и SDA переопределим на лапки портов PB8 и PB9. Делается это нажатием левой кнопки мыши на ножку с зажатой клавишей Ctrl и дальнейшим переносом ее на возможную лапку (нажмите на картинку для увеличения изображения)



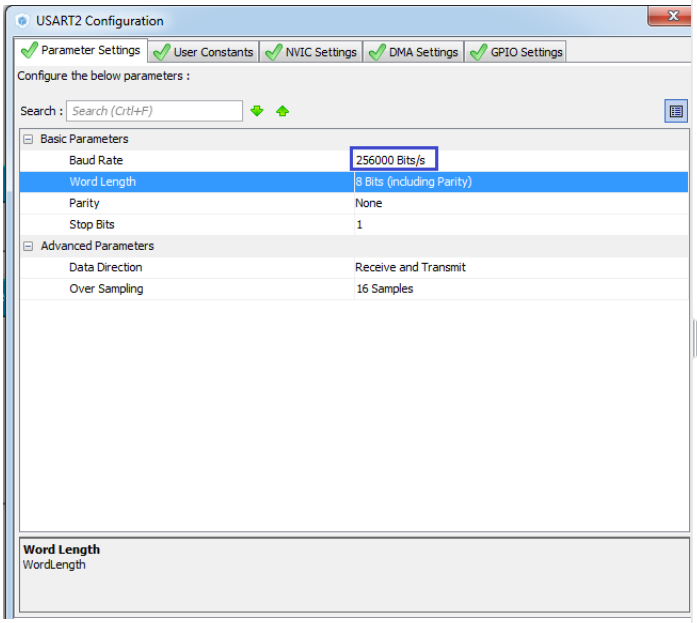
PA13 также отключим

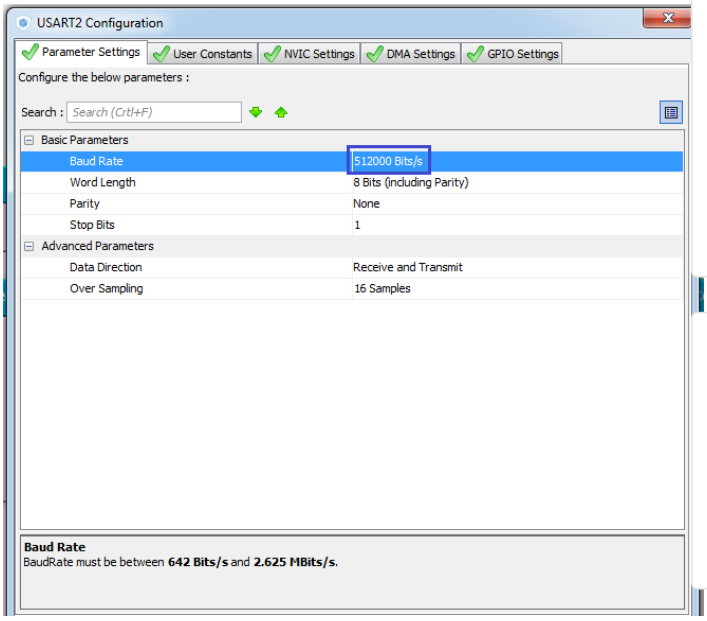


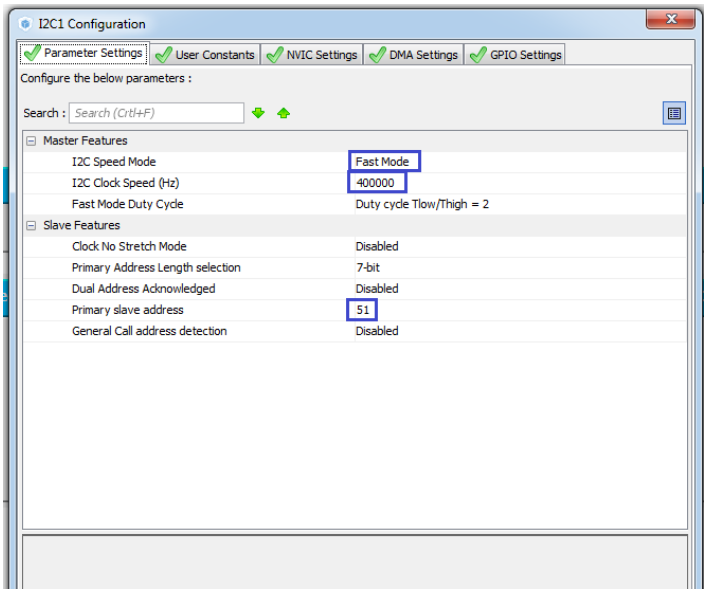
В Clock Configuration ничего не трогаем. В Configuration I2C настроим на 400 кГц, а USART на 256000 bps и включим на USART прерывания и DMA.

Свежие комментарии

- Phoenix75 к записи STM Урок 26. HAL. SPI. Драйвер индикатора MAX7219
- Илья к записи STM Урок 37. Дисплей TFT 240×320 8bit. Часть 4
- admin к записи STM Урок 18. HAL. ADC. Regular Channel. DMA
- ano60 к записи STM Урок 18. HAL. ADC. Regular Channel. DMA



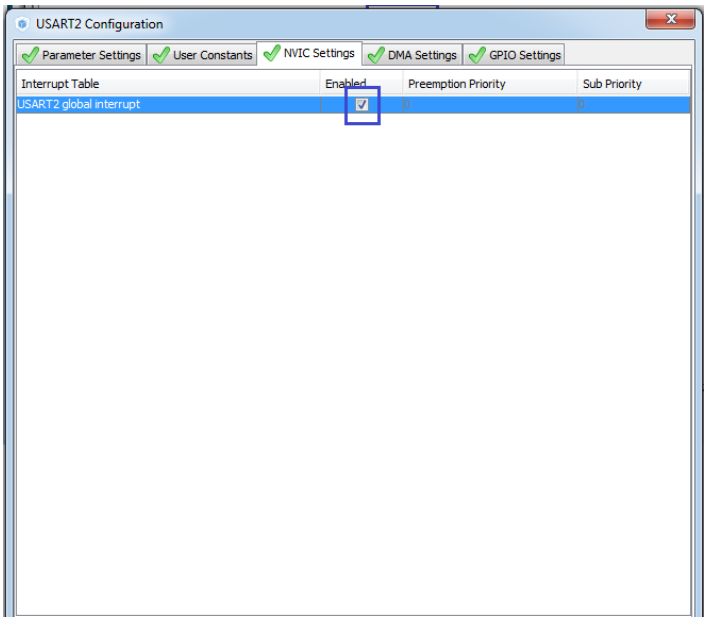


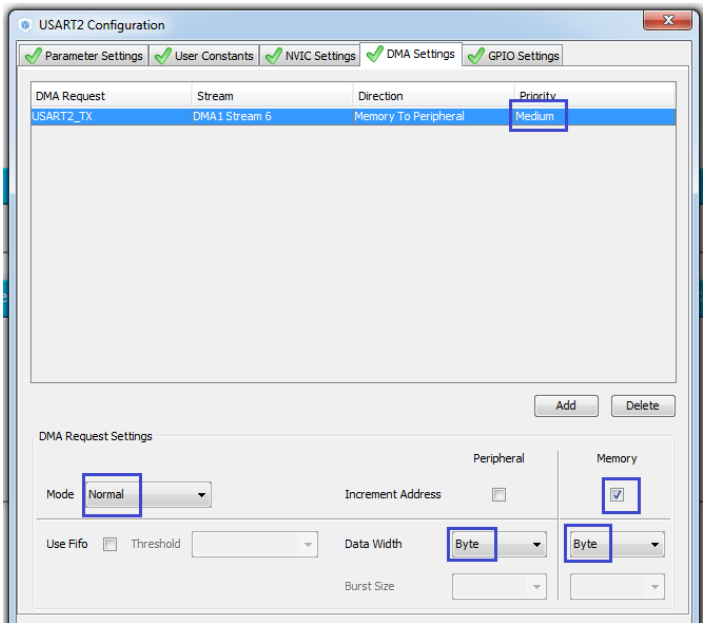


- admin к записи STM Урок 87. LAN. ENC28J60. TCP WEB Server. Передаём страницу побольше. Часть 1

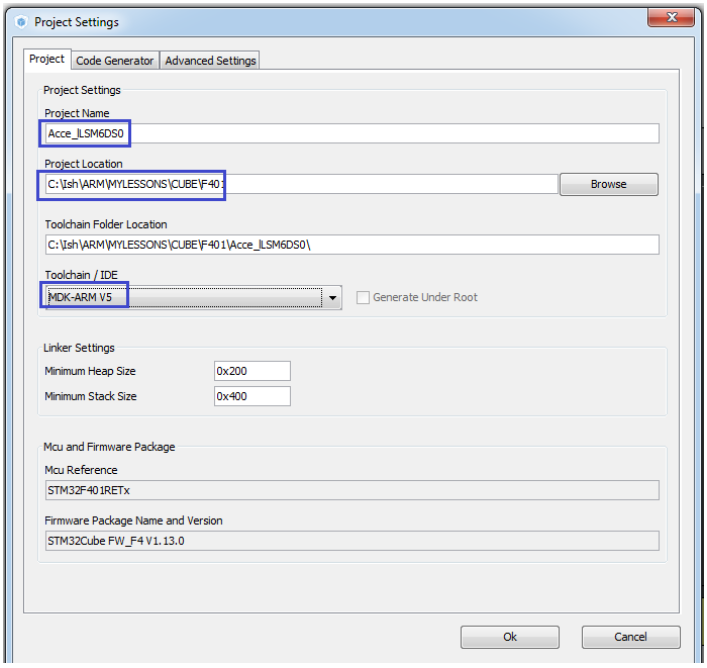
Архивы

- Сентябрь 2017
- Август 2017
- Июль 2017
- Июнь 2017
- Май 2017
- Апрель 2017
- Март 2017
- Февраль 2017
- Январь 2017
- Декабрь 2016
- Ноябрь 2016





Настроим Project Settings, задав имя Acce_lLSM6DS0, среду программирования и путь к проекту. Путь у всех может быть разный.



Сгенерируем проект, откроем его. Настроим программатор на авторезет. Скомпилируем проект.

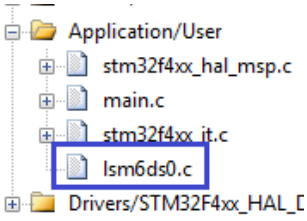
Для дальнейшего составления проекта мы можем воспользоваться опытом занятий с предыдущими датчиками. Файлы библиотек функций датчика мы возьмем с Вами с 39 урока, когда мы подключали датчик, установленный на плате STM32F303 Discovery, так как там мы также пользовались шинами I2C и USART.

В папку Inc проекта мы скопируем файл main.h. Также в соответствующие папки Inc и Src мы скопируем файлы lsm303dlh.h lsm303dlh.c, соответственно переименовав их согласно наименованию нашего датчика в lsm6ds0.h и lsm6ds0.c.

Подключим файл main.h в файле main.c

```
/* USER CODE BEGIN Includes */
#include "main.h"
/* USER CODE END Includes */
```

Подключим также файл lsm6ds0.c



Файл main.h после некоторых исправлений станет вот такого содержания:

```
#ifndef MAIN_H_
#define MAIN_H_

#include "stm32f4xx_hal.h"
#include "lsm6ds0.h"

#endif /* MAIN_H_ */
```

Также подправим заголовочный файл lsm6ds0.h и уберем из него пока лишние макросы

```
#ifndef LIS3DSH_H_
#define LIS3DSH_H_

#include "stm32f4xx_hal.h"
#include <string.h>
//-----------------------------------
#define ABS(x)         (x < 0) ? (-x) : x
//-----------------------------------
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define LD2_ON HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET) //GREEN
#define LD2_OFF HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET)
//-----------------------------------
void Accel_Ini(void);
void Accel_ReadAcc(void);
//-----------------------------------
#endif /* LIS3DSH_H_ */
```

Также подправим и уберем лишнее в файле lsm6ds0.c

```
#include "lsm6ds0.h"
//-----------------------------------
extern I2C_HandleTypeDef hi2c1;
extern UART_HandleTypeDef huart2;
uint8_t buf2[14]={0};
char str1[30]={0};
//-----------------------------------
void Error(void)
{
	LD2_OFF;
}
//-----------------------------------
static uint8_t I2Cx_ReadData(uint16_t Addr, uint8_t Reg)
{
	HAL_StatusTypeDef status = HAL_OK;
	uint8_t value = 0;
	status = HAL_I2C_Mem_Read(&hi2c1, Addr, Reg, I2C_MEMADD_SIZE_8BIT, &value, 1, 0x10000);
	if(status != HAL_OK) Error();
	return value;
}
//-----------------------------------
static void I2Cx_WriteData(uint16_t Addr, uint8_t Reg, uint8_t Value)
{
	HAL_StatusTypeDef status = HAL_OK;
	status = HAL_I2C_Mem_Write(&hi2c1, Addr, (uint16_t)Reg, I2C_MEMADD_SIZE_8BIT, &Value, 1, 0x10000);
	if(status != HAL_OK) Error();
}
//-----------------------------------
uint8_t Accel_IO_Read(uint16_t DeviceAddr, uint8_t RegisterAddr)
{
	return I2Cx_ReadData(DeviceAddr, RegisterAddr);
}
//-----------------------------------
void Accel_IO_Write(uint16_t DeviceAddr, uint8_t RegisterAddr, uint8_t Value)
{
	I2Cx_WriteData(DeviceAddr, RegisterAddr, Value);
}
//-----------------------------------
void Accel_GetXYZ(int16_t* pData)
{
	uint8_t buffer[6];
	uint8_t i=0;
```

```
        for(i=0;i<3;i++)
        {

        }
    }
    //---------------------------------------
    uint8_t Accel_ReadID(void)
    {
        uint8_t ctrl = 0x00;
        return ctrl;
    }
    //---------------------------------------
    void Accel_ReadAcc(void)
    {
        int16_t buffer[3] = {0};
        int16_t xval, yval, zval;
        Accel_GetXYZ(buffer);
        xval=buffer[0];
        yval=buffer[1];
        zval=buffer[2];
    //      sprintf(str1,"X:%06d Y:%06d Z:%06d\r\n", xval,
yval, zval);
    //      HAL_UART_Transmit(&huart2,
(uint8_t*)str1,strlen(str1),0x1000);
        buf2[0]=0x12;
        buf2[1]=0x10;
        buf2[2]=(uint8_t)(xval>>8);
        buf2[3]=(uint8_t)xval;
        buf2[4]=0x10;
        buf2[5]=0x10;
        buf2[6]=(uint8_t)(zval>>8);
        buf2[7]=(uint8_t)zval;
        buf2[8]=0x13;
        HAL_UART_Transmit(&huart2,buf2,9,0×1000);
        if(xval>1500)
        {
        }
        HAL_Delay(20);
    }
    //---------------------------------------
    void AccInit(uint16_t InitStruct)
    {
    }
    //---------------------------------------
    void Accel_Ini(void)
    {
        uint16_t ctrl = 0x0000;
        HAL_Delay(1000);
        AccInit(ctrl);
    }
```

В функцию main добавим следующий код. Мы зажжем светодиод заранее, так как он у нас единственный на плате и вызовем функцию инициализации

```
  /* USER CODE BEGIN 2 */
      HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin,
GPIO_PIN_SET);
      Accel_Ini();
  /* USER CODE END 2 */
```

Начнем, как обычно с попытки считать идентификатор датчика. Обратимся к технической документации. Найдем сначала адрес, необходимый для обращения к устройству по I2C. Если ножка SDO у нас соединена с землей, то 1 бит у нас 0, если с питанием, то 1. 0-й бит у нас будет 0, т.к. мы обращаемся сначала в режиме записи, если потребуется режим воспроизведения, то библиотека HAL сама возьмет на себя заботу об установке его в 1. Остается только узнать, куда же у нас подключена ножка SDO. Схему мне найти не удалось. Осталось 2 варианта, либо пробовать разные адреса, либо взять его из примера. Я выбрал 2й путь как более простой. Поэтому у нас адрес будет последний из таблицы 15 даташита — 11010110 (D6h).

Также нам необходимо знать из какого регистра брать идентификатор и какой он должен быть у нашего датчика, чтобы применить в условии операцию сравнения. Данная информация находится на странице 38.

7.10 WHO_AM_I (0Fh)

Who_AM_I register.

Table 39. WHO_AM_I register

0	1	1	0	1	0	0	0

Поэтому добавим строку в функцию Accel_ReadID

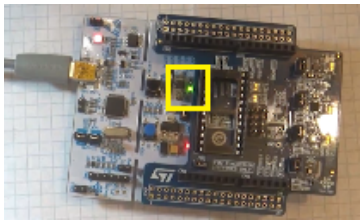
```
uint8_t Accel_ReadID(void)
{
    uint8_t ctrl = 0x00;
    ctrl = Accel_IO_Read(0xD6,0x0F);
    return ctrl;
}
//————————————————————————
```

Также вставим строки в функцию инициализации датчика

```
    HAL_Delay(1000);
    if(Accel_ReadID()==0x68) LD2_ON;
    else Error();
    AccInit(ctrl);
```

Соберем код и прошьем контроллер и проверим тем самым, тот ли у нас датчик, сравнив с образцовым идентификатором и включив единственный зеленый светодиод. Но так как светодиод у нас уже зажжен был на этапе старта программы, то нам теперь главное, чтобы она не погас.

Если светодиод у нас остался гореть, значит у нас все нормально и можно дальше продолжить писать код инициализации акселерометра



Добавим переменную в функцию AccInit

```
void AccInit(uint16_t InitStruct)
{
  uint8_t value=0;
```

Добавим некоторые макросы в заголовочный файл lsm6ds0.h, скопировав их из заранее подготовленного файла

```
#define LD2_OFF HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET)
//————————————————————————
#define LSM6DS0_ACC_GYRO_BDU_DISABLE  0x00
#define LSM6DS0_ACC_GYRO_BDU_ENABLE   0x40
#define LSM6DS0_ACC_GYRO_BDU_MASK   0x40
//————————————————————————
#define LSM6DS0_ACC_GYRO_CTRL_REG5_XL 0X1F
#define LSM6DS0_ACC_GYRO_CTRL_REG6_XL 0X20
```

```
    #define LSM6DS0_ACC_GYRO_CTRL_REG8    0X22
    //------------------------------------
    #define
LSM6DS0_ACC_GYRO_ODR_XL_POWER_DOWN 0x00
    #define LSM6DS0_ACC_GYRO_ODR_XL_10Hz 0x20
    #define LSM6DS0_ACC_GYRO_ODR_XL_50Hz 0x40
    #define LSM6DS0_ACC_GYRO_ODR_XL_119Hz 0x60
    #define LSM6DS0_ACC_GYRO_ODR_XL_238Hz 0x80
    #define LSM6DS0_ACC_GYRO_ODR_XL_476Hz 0xA0
    #define LSM6DS0_ACC_GYRO_ODR_XL_952Hz 0xC0
    //------------------------------------
    #define LSM6DS0_ACC_GYRO_ODR_XL_MASK
 0xE0
    //------------------------------------
    #define LSM6DS0_ACC_GYRO_FS_XL_2g 0x00
    #define LSM6DS0_ACC_GYRO_FS_XL_16g 0x08
    #define LSM6DS0_ACC_GYRO_FS_XL_4g 0x10
    #define LSM6DS0_ACC_GYRO_FS_XL_8g 0x18
    //------------------------------------
    #define LSM6DS0_ACC_GYRO_FS_XL_MASK   0x18
    //------------------------------------
    #define LSM6DS0_ACC_GYRO_XEN_XL_ENABLE
0x08
    #define LSM6DS0_ACC_GYRO_YEN_XL_ENABLE
0x10
    #define LSM6DS0_ACC_GYRO_ZEN_XL_ENABLE
0x20
    #define LSM6DS0_ACC_GYRO_XEN_XL_MASK
0x08
    #define LSM6DS0_ACC_GYRO_YEN_XL_MASK
0x10
    #define LSM6DS0_ACC_GYRO_ZEN_XL_MASK
0x20
    //------------------------------------
    #define LSM6DS0_ACC_GYRO_OUT_X_L_XL   0X28
    #define LSM6DS0_ACC_GYRO_OUT_X_H_XL   0X29
    #define LSM6DS0_ACC_GYRO_OUT_Y_L_XL   0X2A
    #define LSM6DS0_ACC_GYRO_OUT_Y_H_XL   0X2B
    #define LSM6DS0_ACC_GYRO_OUT_Z_L_XL   0X2C
    #define LSM6DS0_ACC_GYRO_OUT_Z_H_XL   0X2D
    //------------------------------------
    void Accel_Ini(void);
```

Что означают данные настройки и регистры, мы будем разбирать по мере написания функций.

В **следующей части** занятия мы продолжим писать функцию инициализации датчика, затем напишем функцию считывания данных, и, более того, мы в данной части закончим работу с данным акселерометром.
Мы увидим показания и в текстовом виде и визуально с помощью программы NS Port Monitor.

Предыдущий урок
Программирование МК STM32
Следующая часть

Техническая документация на датчик
Техническая документация на плату расширения

Программа NS Port Monitor

Смотреть ВИДЕОУРОК