ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

РАБОТА С АКСЕЛЕРОМЕТРОМ

по курсу: ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № _____ _____ _____
подпись, дата инициалы, фамилия

Санкт-Петербург 2020

Цель работы

Изучить работу акселерометра на примере управления акселерометром LIS302DL

1 Теоретические сведения

Акселерометр – это измерительный прибор позволяющий определить проекцию кажущегося ускорения. В простейшем исполнении он представляет собой грузик, закрепленный на упругом подвесе. При его отклонении от первоначального положения на упругом подвесе можно определить направление изменения положения, а также величину ускорения.

На плате stm32f4Discovery она подключена к выходам, показанным на рисунке 1

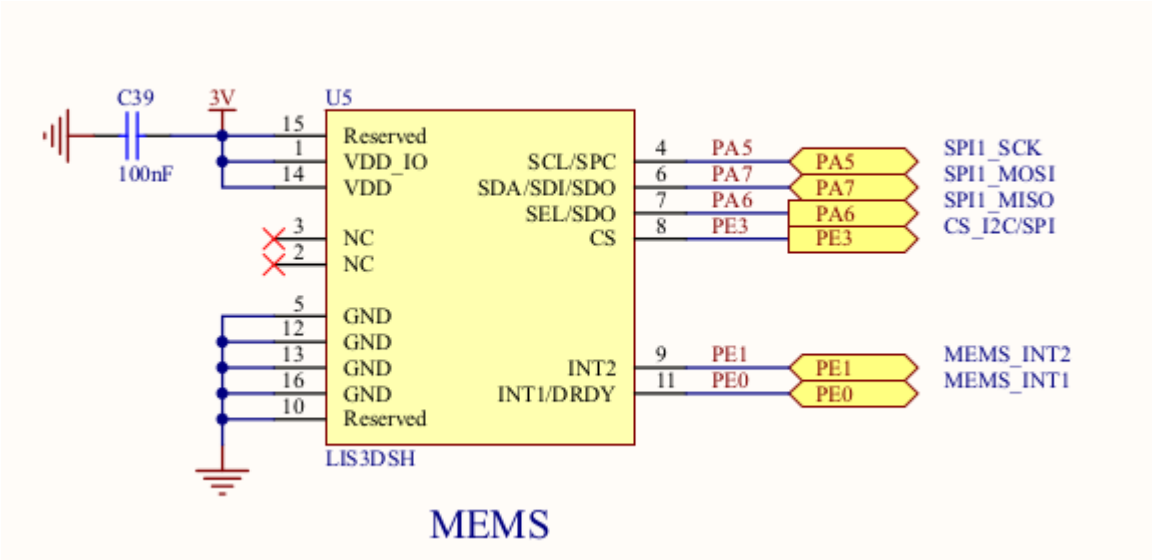

Рисунок 1 – Схема подключения портов STM32F407VG к LIS302DL

Сам акселерометр обладает следующими характеристиками:

- Напряжение питания 2.16 — 3.6 в.
- Измерение ускорения по трём осям
- Два диапазона измерения 2G/8G
- Два настраиваемых выхода для прерываний
- Самодиагностика
- Обнаружение кликов (постукиваний)
- Встроенный фильтр
- Корпус LGA14

Взаимодействие с акселерометром осуществляется через его

регистры, ознакомиться с ними можно в его даташите. Чтобы прочитать или что-то записать в них, нужно отправить через SPI посылку определённого формата. Рассмотрим способ записи данных в акселерометр, показанный на рисунке 2

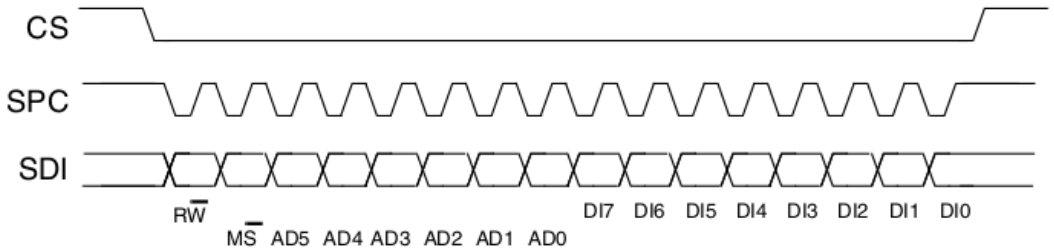

Рисунок 2 – Запись данных в акселерометр

Где:

- **DO7**..**DO0** — байт данных отправленный акселерометром в микроконтроллер
- **DI7**..**DI0** — байт данных переданный микроконтроллером в акселерометр
- **AD5**..**AD0** — адрес записываемого/считываемого регистра
- **RW** — если бит=0 то байт данных **DI7**..**DI0** будет записан в регистр по адресу **AD5**..**AD0,** в противном случае (RW=1) байт данных **DO7**..**DO0** будет прочитан из регистра по адресу **AD5**..**AD0**
- **MS** — используется если мы хотим прочитать/записать несколько регистров подряд. Если бит сброшен, то после передачи адреса мы будем считывать/записывать один и тот же регистр вне зависимости от того, сколько раз будет предпринята попытка чтения/записи. Если же этот бит установлен, то адрес будет автоматически увеличиваться на единицу после каждой записи или чтения.

2 Практическая часть

В рамках работы была написана программа с помощью библиотеки CMSIS

Файл “main.c”

```
#include "main.h"

void Delay(volatile uint32_t nCount)
{
	while(nCount--) {}
}

int main(void){
	int8_t Accel_x, Accel_y;

	// Инициализируемуем требуемую перефирию
	LEDs_ini();
	TIM4_ini();
	SPI1_ini();

	// Включаем отслеживание по осям y и x, а так же включаем сам акселерометр
	SPI_Tx(CTRL_REG1, CTRL_REG1_PD | CTRL_REG1_XEN | CTRL_REG1_YEN);

	while(1){
		// Считываем измеренные акселерометром параметры
		Accel_x = SPI_Rx(OUTX);
		Accel_y = SPI_Rx(OUTY);

		// В зависимости от полярности параметров меняем скважность ШИМ
		if (Accel_x < -NOISE_LIMIT || Accel_x > NOISE_LIMIT){
			if (Accel_x < 0){
				WRITE_REG(TIM4->CCR1, -Accel_x);
				WRITE_REG(TIM4->CCR3, 0);
			}
			else{
				WRITE_REG(TIM4->CCR3, Accel_x);
				WRITE_REG(TIM4->CCR1, 0);
			}
		}
		if (Accel_y < -NOISE_LIMIT || Accel_y > NOISE_LIMIT){
			if (Accel_y < 0){
				WRITE_REG(TIM4->CCR4, -Accel_y);
				WRITE_REG(TIM4->CCR2, 0);
			}
			else{
```

```
                        WRITE_REG(TIM4->CCR2, Accel_y);
                        WRITE_REG(TIM4->CCR4, 0);
                    }
                }
            }
}
```

Файл “main.h”

```
#include "init.h"
#include "spi.h"

#define NOISE_LIMIT 4

#ifndef MAIN_H
#define MAIN_H
//
#endif
```

Файл “init.c”

```
#include "init.h"

void LEDs_ini(void){
    SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIODEN);
    SET_BIT(GPIOD->MODER, GPIO_MODER_MODER12_1 | GPIO_MODER_MODER13_1 | GPIO_MODER_MODER14_1 | GPIO_MODER_MODER15_1); // Установка портов в альтернативный режим работы
    WRITE_REG(GPIOD->AFR[1], 0x22220000); // Подключение вывода 4-х каналов таймера 4 к 4-ом диодам
}

void TIM4_ini(void){
    SET_BIT(RCC->APB1ENR, RCC_APB1ENR_TIM4EN);
    SET_BIT(TIM4->CCER, TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC3E | TIM_CCER_CC4E); // Разрешение использовать порты ввода-вывода к которым подключены каналы таймера для ШИМа
    WRITE_REG(TIM4->ARR, 0xFF); // Установка предела счёта таймера
    // Выставляем на всех каналах таймера 4 режим  работы – ШИМ
    SET_BIT(TIM4->CCMR1, TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);
```

```
	SET_BIT(TIM4->CCMR1, TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2);
	SET_BIT(TIM4->CCMR2, TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2);
	SET_BIT(TIM4->CCMR2, TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2);
	SET_BIT(TIM4->CR1, TIM_CR1_CEN | TIM_CR1_ARPE ); // Включаем таймер и заставляем его запоминать значение предела счёта таймера
}

void SPI1_ini(void){
	SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN);
	SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOEEN);

	SET_BIT(GPIOA->MODER, GPIO_MODER_MODER5_1 | GPIO_MODER_MODER6_1 | GPIO_MODER_MODER7_1); // Установка портов в альтернативный режим работы
	SET_BIT(GPIOA->AFR[0], 0x55500000); // Подключение выводов SPI модуля к 3-м портам

	// Настройка порта отвечающего за сигнал о начале и окончании передачи данных
	SET_BIT(GPIOE->MODER, GPIO_MODER_MODER3_0);
	SET_BIT(GPIOE->PUPDR, GPIO_PUPDR_PUPDR3_0);
	// Сигнализируем, что передача данных не ведётся
	SET_BIT(GPIOE->BSRRL, GPIO_ODR_ODR_3);

	SET_BIT(RCC->APB2ENR, RCC_APB2ENR_SPI1EN);

	// Настройка SPI для работы с акселерометром
	WRITE_REG(SPI1->CR1, SPI_CR1_SSM | SPI_CR1_SSI | SPI_CR1_MSTR | SPI_CR1_CPHA | SPI_CR1_CPOL | SPI_CR1_BR_1 | SPI_CR1_BR_2);
	// Включение SPI
	SET_BIT(SPI1->CR1, SPI_CR1_SPE);
}
```

Файл “init.h”

```
#include "stm32f4xx.h"

#define PLL_M 8
```

```
#define PLL_N                   336
#define PLL_P                   2
#define PLL_Q                   7
#define AHB_PRE                 RCC_CFGR_HPRE_DIV1
#define APB1_PRE            RCC_CFGR_PPRE1_DIV4
#define APB2_PRE            RCC_CFGR_PPRE2_DIV2

void LEDs_ini(void);
void TIM4_ini(void);
void SPI1_ini(void);
```

Файл “spi.c”

```
#include "spi.h"

void SPI_Tx(uint8_t address, uint8_t data){
	// Сигнализируем о начале передачи
	SET_BIT(GPIOE->BSRRH, GPIO_ODR_ODR_3);
	// Ждём готовности передачи
	while (READ_BIT(SPI1->SR, SPI_SR_TXE) == 0);
	// Передаём адрес регистра в который будем записывать данные
	WRITE_REG(SPI1->DR, address);
	// Ждём ответа устройства
	while (READ_BIT(SPI1->SR, SPI_SR_RXNE) == 0);
	// Считываем мусор, чтобы была возможность послать следующее
сообщение
	READ_REG(SPI1->DR);
	while (READ_BIT(SPI1->SR, SPI_SR_TXE) == 0);
	// Передаём данные, которые нужно записать
	WRITE_REG(SPI1->DR, data);
	while (READ_BIT(SPI1->SR, SPI_SR_RXNE) == 0);
	// Считываем мусор
	READ_REG(SPI1->DR);
	// Ждём окончания работы SPI
	while (READ_BIT(SPI1->SR, SPI_SR_BSY) == SPI_SR_BSY);
	// Сигнализируем об окончании передачи
	SET_BIT(GPIOE->BSRRL, GPIO_ODR_ODR_3);
}

uint8_t SPI_Rx(uint8_t address){
	uint8_t recived_byte;

	// Сигнализируем о начале передачи
	SET_BIT(GPIOE->BSRRH, GPIO_ODR_ODR_3);
	// Ждём готовности передачи
```

```
	while (READ_BIT(SPI1->SR, SPI_SR_TXE) == 0);
	// Передаём бит чтения данных и адрес регистра, который мы собирамся прочесть
	WRITE_REG(SPI1->DR, address | TRANSMIT_RW);
	// Ждём ответа устройства
	while (READ_BIT(SPI1->SR, SPI_SR_RXNE) == 0);
	// Считываем мусор, чтобы была возможность принять новые данные
	recived_byte = READ_REG(SPI1->DR);
	while (READ_BIT(SPI1->SR, SPI_SR_TXE) == 0);
	// Посылаем мусор, чтобы устройство послало нам запрашиваемые данные
	WRITE_REG(SPI1->DR, 0x00);
	while (READ_BIT(SPI1->SR, SPI_SR_RXNE) == 0);
	// Наконец считаываем нужное нам
	recived_byte = READ_REG(SPI1->DR);
	// Ждём окончания работы SPI
	while (READ_BIT(SPI1->SR, SPI_SR_BSY) == SPI_SR_BSY);
	// Сигнализируем об окончании передачи
	SET_BIT(GPIOE->BSRRL, GPIO_ODR_ODR_3);

	return recived_byte;
}
```

Файл “spi.h”

```
#include "stm32f4xx.h"

#define CTRL_REG1
                        ((uint8_t) 0x20)
#define OUTX
                                ((uint8_t) 0x29)
#define OUTY
                                ((uint8_t) 0x2B)

#define CTRL_REG1_XEN
                        ((uint8_t) 0x01)
#define CTRL_REG1_YEN
                        ((uint8_t) 0x02)
#define CTRL_REG1_FS
                        ((uint8_t) 0x20)
#define CTRL_REG1_PD
                        ((uint8_t) 0x40)
#define CTRL_REG1_DR
                        ((uint8_t) 0x80)
```

```
#define TRANSMIT_RW
                                ((uint8_t) 0x80)

void SPI_Tx(uint8_t address, uint8_t data);
uint8_t SPI_Rx(uint8_t address);
```

3 Результаты работы программы.

Вывод компилятора показан на рисунке 3

Работу программы можно увидеть по ссылке https://imgur.com/a/l9CDx8U

```
Rebuild started: Project: lab7
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Rebuild target 'lab7'
assembling startup_stm32f40xx.s...
compiling main.c...
compiling spi.c...
compiling system_stm32f4xx.c...
compiling init.c...
linking...
Program Size: Code=1344 RO-data=408 RW-data=0 ZI-data=1632
FromELF: creating hex file...
".\Objects\sample_project.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

Рисунок 3 – Компиляция программы (build output)

Вывод

В рамках данной лабораторной работы мной был написан код на языке C, с использованием библиотеки CMSIS, для stm32f4, который взаимодействует с акселерометром с помощью spi протокола, считывает значения по x и y осям акселерометра, которые становятся значением скважности диодов, отражая изменение значений.