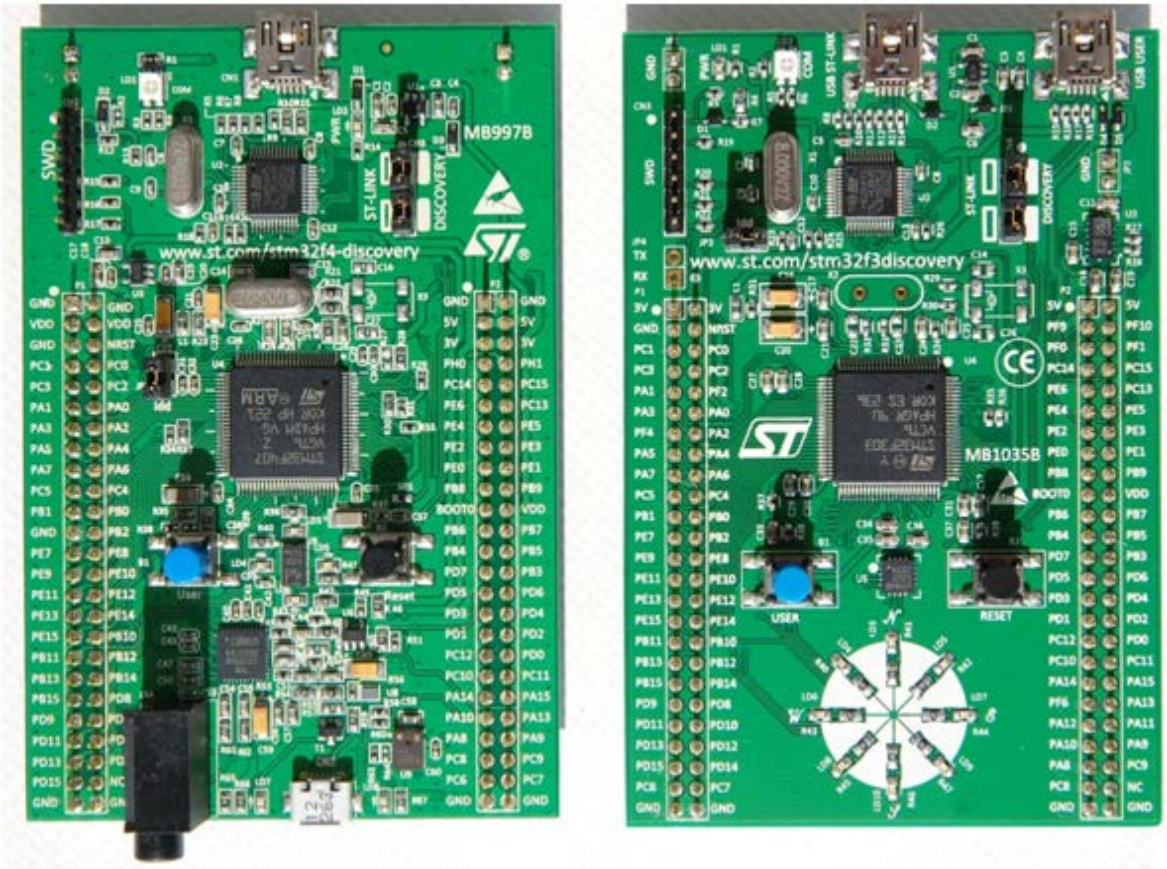
В. И. Бугаев, М. П. Мусиенко, Я. М. Крайнык

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по изучению микроконтроллеров STM32 на базе отладочных модулей STM32F3 Discovery и STM32F4 Discovery



Москва-Николаев – 2014

Лабораторный практикум по изучению микроконтроллеров STM32 на базе отладочного модуля STM32F3 Discovery и STM32F4 Discovery / Бугаев В.И., Мусиенко М.П., Крайнык Я.М. – Москва-Николаев: МФТИ-ЧГУ, 2014. – 26 с.

Бугаев Виктор Иванович, Московский физико-технический институт, г. Москва, Россия

Мусиенко Максим Павлович, д.т.н., профессор, Черноморский государственный университет им. П. Могилы, г. Николаев, Украина

Крайнык Ярослав Михайлович, аспирант, Черноморский государственный университет им. П. Могилы, г. Николаев, Украина

СОДЕРЖАНИЕ

Лабораторная работа №1. Изучение интерфейса SPI в STM32F3 Discovery

Цель работы: исследовать возможности использования интерфейса SPI для передачи данных.

Оборудование и программное обеспечение: отладочная плата STM32F3 Discovery, логический анализатор на базе STM32F4 Discovery (см. Приложение А), среда разработки Keil uVision.

Теоретические сведения

Serial Peripheral Interface (SPI) – простой последовательный двунаправленный интерфейс. Интерфейс является синхронным: передача и прием выполняются при наличии тактового сигнала. Тактовый сигнал генерируется ведущим устройством, которое передает сообщение. Главным составляющим блоком интерфейса SPI является простой регистр сдвига, сигналы синхронизации ввода/вывода битового потока которого и являются интерфейсными сигналами. Таким образом, протокол SPI можно называть протоколом обмена сообщениями между двумя регистрами сдвига, каждый из которых одновременно выполняет функцию передатчика и приемника. Неотъемлемым условием передачи данных по шине SPI является генерация сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущее устройство на шине и от него полностью зависит работа ведомых устройств.

Одноименные выводы двух устройств соединяются между собой при подключении. Присутствуют два канала передачи данных (MOSI и MISO), один канал для передачи тактовых импульсов (SCLK), а также линия включения ведомого устройства (SS). Ведомый стает активным при подаче сигнала низкого уровня по линии SS.

Ход работы

Далее проведем изучение сигналов, которые используются для передачи данных упомянутым интерфейсом с помощью отладочного модуля STM32F3 Discovery. Структура проекта в среде Keil представлена на рисунке 1.

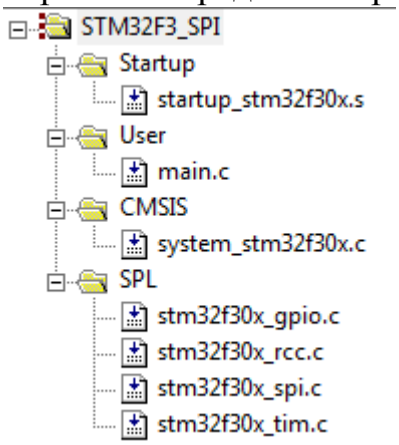


Рис. 1. Структура проекта

В данном примере будем передавать строку с помощью интерфейса SPI и просматривать результат передачи с помощью цифрового анализатора. Далее приведена программная реализация для проведения данных действий.

```
#include "stm32f30x.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_rcc.h"
#include "stm32f30x_spi.h"
#include "stm32f30x_tim.h"

void init(void);
void spi_send_char(char data);
void spi_send_string(char * string);

void TIM2_IRQHandler(void) {
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    spi_send_string("stm spi");
}

int main() {
     __enable_irq();
    init();
    NVIC_EnableIRQ(TIM2_IRQn);
    while(1) {
    }
}

void init(void) {
    GPIO_InitTypeDef gpio_init;
    SPI_InitTypeDef spi_init;
    TIM_TimeBaseInitTypeDef tim_init;

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    GPIO_StructInit(&gpio_init);
    gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    gpio_init.GPIO_Mode = GPIO_Mode_AF;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &gpio_init);

    GPIO_PinAFConfig(GPIOB, GPIO_PinSource12, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_5);

    SPI_I2S_DeInit(SPI2);
    spi_init.SPI_Mode = SPI_Mode_Master;
    spi_init.SPI_Direction = SPI_Direction_Tx;
    spi_init.SPI_CPOL = SPI_CPOL_Low;
    spi_init.SPI_CPHA = SPI_CPHA_1Edge;
    spi_init.SPI_NSS = SPI_NSS_Soft;
```

```
    spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
    spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
    spi_init.SPI_CRCPolynomial = 7;
    SPI_CalculateCRC(SPI2, DISABLE);
    SPI_Init(SPI2, &spi_init);
    SPI_DataSizeConfig(SPI2, SPI_DataSize_8b);
    SPI_Cmd(SPI2, ENABLE);

    TIM_TimeBaseStructInit(&tim_init);
    tim_init.TIM_Period = 16000;
    tim_init.TIM_Prescaler = 100;
    TIM_TimeBaseInit(TIM2, &tim_init);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

    TIM_Cmd(TIM2, ENABLE);
}

void spi_send_char(char data) {
    while(!(SPI2->SR & SPI_SR_TXE));
    SPI_SendData8(SPI2, data);
}

void spi_send_string(char * string) {
    uint8_t i = 0;
    while (string[i]) {
        spi_send_char(string[i]);
        i++;
    }
}
```

Проверку проведем с помощью логического анализатора (см. Приложение А) с функцией анализатора интерфейса SPI. Результаты показали, что строка передается корректно (рис. 2). Важно правильно указать каналы SPI, которые подключены к входам логического анализатора.

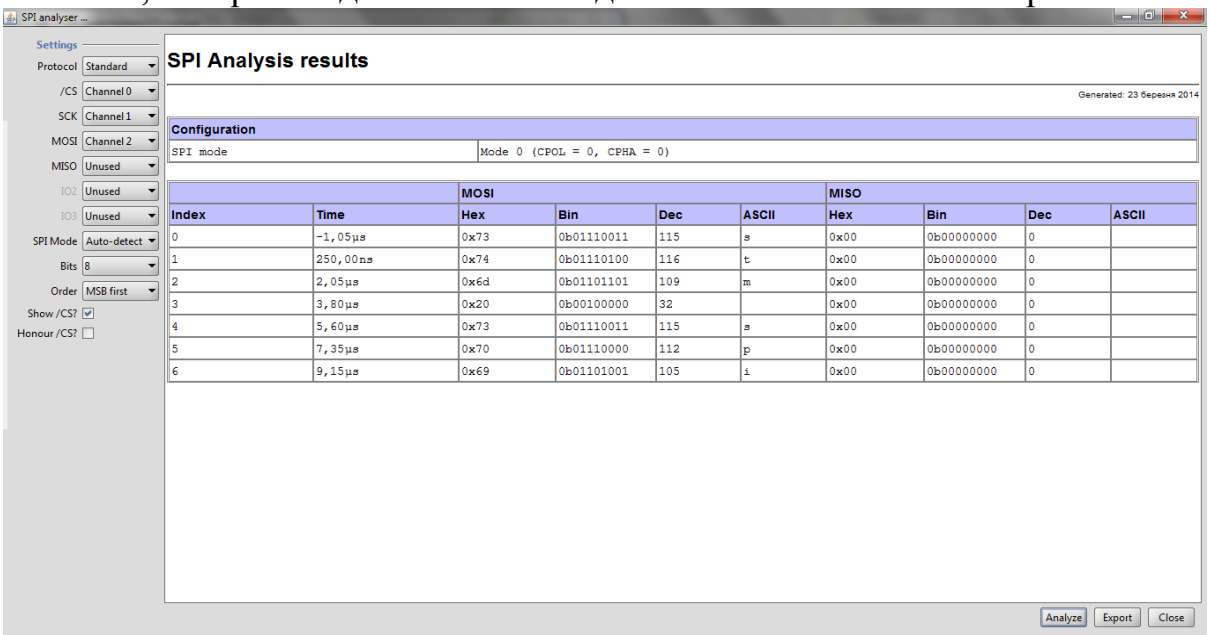


Рис. 2. Результаты, полученные анализатором протокола SPI

Есть возможность просмотреть вид сигналов, которые передаются с помощью программного обеспечения (рис. 3; сигнал на линии SCK на рисунке соответствует MOSI).

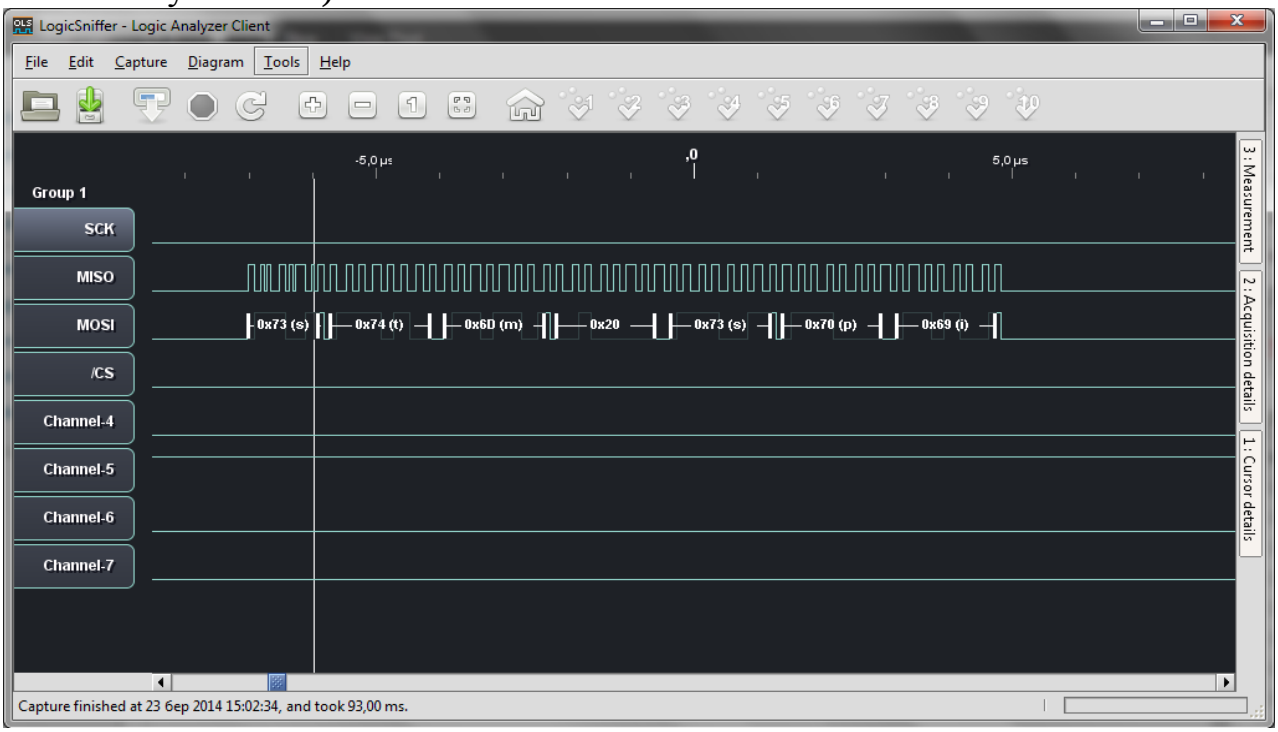


Рис. 3. Диаграммы значений цифровых сигналов

Как видим, с помощью логического анализатора удобно изучать интерфейсы передачи данных.

Задание

1. Проверить работу программы, приведенной в примере. Получить значения сигналов, как указаны на рисунке.
2. Организовать работу между двумя модулями SPI, которые присутствуют в составе микроконтроллера. Просмотреть с помощью логического анализатора данные, которые передаются и получаются в ответ. Протокол обмена можно выбрать самостоятельно.
3. Результаты продемонстрировать в виде графиков, полученных с помощью логического анализатора. Привести соответствие между сигналами на линиях MOSI и MISO.

Лабораторная работа №2. Изучение интерфейса I2C. Работа с EEPROM-памятью

Цель работы: изучение работы с EEPROM-памятью, организацию записи и считывания информации.

Оборудование и программное обеспечение: микросхема памяти BR24C04 или соответствующий аналог, отладочная плата STM32F4 Discovery, перемычки.

Теоретические сведения

I2C (Inter-Integrated Circuit) – последовательный интерфейс передачи данных, который используется для подключения периферийных компонентов. Разработчиком данного интерфейса является фирма Phillips, однако его использует большое количество других производителей. Интерфейс предназначен в основном для обмена данными на низких скоростях. Хотя последняя версия стандарта способна обеспечивать скорость более 3 Мбит/с, но даже этот показатель уступает показателям, например, SPI. Более ранние версии работают на скоростях 100 и 400 кбит/с.

Передача данных происходит с помощью двух линий:

1. Линии тактовой частоты (обычно обозначается SCL).
2. Линии передачи данных (обозначение – SDA).

Обе линии передачи подтянуты к напряжению питания с помощью резисторов, поэтому уровень сигнала по умолчанию при отсутствии прохождения передачи соответствует логической единице. Линии являются двунаправленными, используется управление уровнями на основе вывода типа открытый сток. С помощью двух линий можно подключить более 1 устройства, что является возможным благодаря протоколу передачи данных. Различают устройство, которое инициирует обмен данными, ведущий, а также ведомое устройство. Таких устройств к линии передачи можно подключить более 1 – это определяется диапазоном адресов, которые могут принимать микросхемы при подключении. Может быть несколько как ведущих, так и ведомых.

То, как именно ведется обмен данными, зависит от конкретной микросхемы, это процесс может отличаться. Однако существуют определенные общепринятые и стандартизированные последовательности сигналов, которых придерживаются при использовании. Во-первых, это последовательности начала и завершения передачи, сигналы START и STOP. Устройства распознают эти последовательности при высоком логическом уровне на линии SCL. В случае сигнала START значение на SDA меняется из высокого на низкий уровень, а для STOP – с низкого на высокий. Все эти действия происходят при значении лог. 1 на линии SCL. При передаче данных изменение логического уровня на линии SDA разрешается только при лог. 0 на линии SCL. Количество бит, которое передается, – 8. После завершения передачи битов ведущий может получить сигнал о подтверждении приема – сигнал ACK (от Acknowledgement). Это проявляется в том, что он отпускает линию SDA, а приемник притягивает ее к уровню лог. 0 на 9 тактовом импульсе передачи. В случае передачи более 1 байта та кие действия повторяются. В случае, когда ведущий выступает в роли приемника, возможен сигнал NACK, который проявляется в том, что после приема последнего байта он выставляет не лог. 0, а лог. 1 на линии SDA, что свидетельствует о завершении передачи. За этим следует сигнал STOP. Также следует отметить сигнал повторного старта, который используется для того, чтобы изменить направление передачи информации. Сигнал повторного старта имеет место, когда обмен информацией уже начат.

Большое количество устройств использует данный интерфейс для обмена данными, например, его можно встретить на материнских платах в компьютерах, в микросхемах телевизоров, часто используют датчики. Едва ли не наибольшее распространение I2C приобрел в микросхемах памяти EEPROM. Такие микросхемы содержат небольшой объем памяти (от 128 байт до 32 кбайт и более) и используются для хранения определенных данных. Например, известно, что именно такие микросхемы используются в принтерах. После выключения питания содержимое памяти сохраняется и его можно прочитать, подав питание на микросхему.

Ход работы

В ходе выполнения данной лабораторной работы предлагается рассмотреть использование микросхемы EEPROM-памяти BR24C04, выполнение записи и считывания информации из нее. Микросхема содержит 512 байт памяти, которые размещены в 2 страницы по 256 байт. Такого объема достаточно, например, для сохранения настроек устройства. Рассмотрим детальнее работу с микросхемой с помощью интерфейса I2C. Воспользуемся диаграммами, приведенными в документации.

Рассмотрим выполнение записи в память по определенному адресу. График данного процесса для линии SDA представлен на рис. 4.

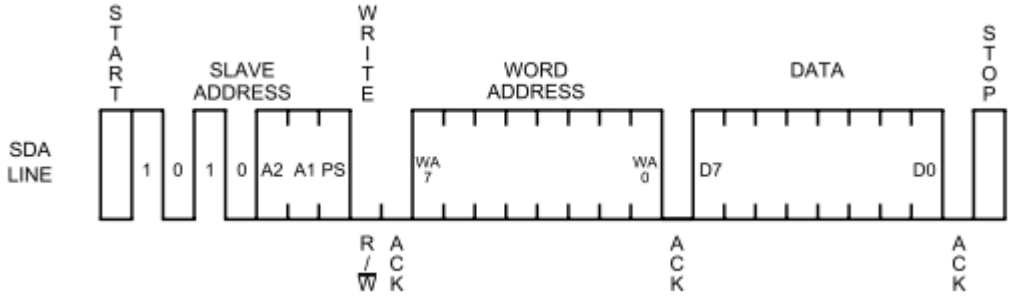


Рис. 4. Последовательность выполнения записи данных

Для выполнения записи следует сгенерировать сигнал START. Далее первое, на что следует обратить внимание, – фиксированные биты адреса для микросхемы. Первые четыре бита не могут изменяться. Вслед за фиксированной последовательность следует передать адрес устройства (биты A1 и A2), в соответствии с логическими уровнями, которые поданы на эти входы. В данном случае эти выводы подключены к земле, поэтому адрес устройства будет 0x00. Таким образом, максимальное количество таких устройств на одной шине составляет 4. За битами адреса следует бит выбора страницы. Последний бит – выбор режима записи (0) или считывания (1). После получения подтверждения (ACK) от микросхемы происходит передача адреса данных. Следующим этапом является передача собственно данных. Все этапы должны сопровождаться получением подтверждения о получении. В конце процесса записи ведущий (в нашем случае отладочная плата) должен сгенерировать сигнал STOP. Следует отметить, что после первого записанного байта можно продолжить запись. При этом текущий адрес инкрементируется на

1 и данные записываются последовательно. После каждого записанного байта микросхема отправляет сигнал подтверждения ACK.

Процесс считывания данных по определенному адресу является более сложным и требует выполнения большего количества шагов (рис. 5).

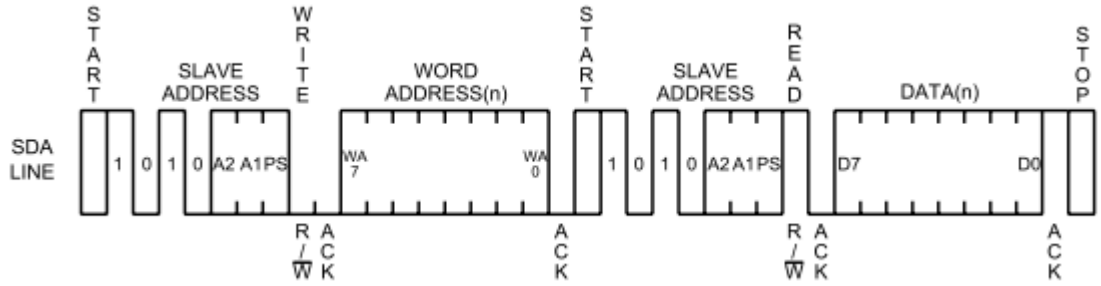
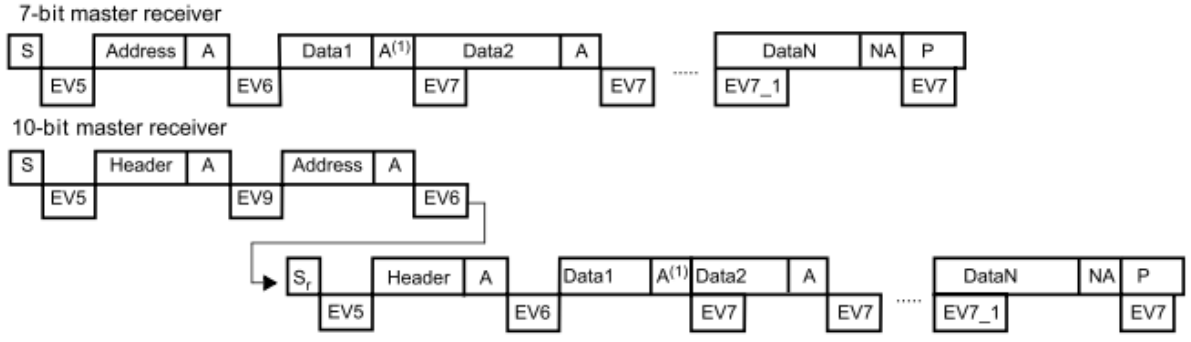


Рис. 5. Последовательность выполнения считывания данных по определенному адресу

Если начальные шаги не отличаются от режима записи, то после отправки адреса и получения подтверждения, следует сгенерировать сигнал повторного старта. После этого отправляется байт адреса, но с учетом того, что необходимо проводить считывание (последний байт выставлен в лог. 1). Отправив сигнал подтверждения, микросхема отправит приемнику данные, которые находятся по указанному адресу. Получив один байт, ведущее устройство выставляет лог. 1 на линии SDA – сигнал NACK. Для завершения обмена генерируется сигнал STOP.

Ознакомившись с тем, как именно проходит процесс обмена со стороны микросхемы, следует также определить, как именно выполняются необходимые действия со стороны микроконтроллера. Документация для устройства приводит диаграммы последовательности выполнения действий (рис. 6) на примере, когда он выступает в качестве ведущего устройства, которое получает информацию (master receiver).



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

EV5: SB=1, cleared by reading SR1 register followed by writing DR register.

EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.

In case of the reception of 1 byte, the Acknowledge disable must be performed during EV6 event, i.e. before clearing ADDR flag.

EV7: RxNE=1 cleared by reading DR register.

EV7_1: RxNE=1 cleared by reading DR register, program ACK=0 and STOP request

EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

Рис. 6. Последовательность считывания для примера микроконтроллера-получателя

Из рисунка и комментариев к нему видно, что генерация необходимых сигналов выполняется установкой битов в управляющем регистре, а проверка событий (они обозначены как EVX) – считыванием регистра статуса и проверкой установленных бит.

Рассмотрим пример записи данных по определенному адресу и их дальнейшим считыванием. Установим адрес начального выполнения операций – 0x10, и количество циклов – 10. В память записываем значение инкремента, который используется для организации цикла. Обратите внимание на адреса, которые используются для начала обмена. Определите, почему именно такие числа используются.

```
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_i2c.h>
#include <stm32f4xx_rcc.h>

#define START_ADDRESS 0x10

void init(void);
void i2c_start(I2C_TypeDef * i2c, uint8_t address);
void i2c_write(I2C_TypeDef * i2c, uint8_t data);
uint8_t i2c_read(I2C_TypeDef * i2c);
void i2c_stop(I2C_TypeDef * i2c);

int main(void)
{
	uint8_t i;
	uint8_t res;

	init();
	i2c_start(I2C1, 160);
	i2c_write(I2C1, START_ADDRESS);
	for (i = 0; i < 10; ++i) {
		i2c_write(I2C1, i);
	}
	i2c_stop(I2C1);

	for (i = 0; i < 10; i++) {
		i2c_start(I2C1, 160);
		i2c_write(I2C1, START_ADDRESS + i);
		i2c_start(I2C1, 161);
		res = i2c_read(I2C1);
		if (res != i) break;
		i2c_stop(I2C1);
	}

    while(1)
    {
    }
}

void init(void) {
	I2C_InitTypeDef i2c_init;
	GPIO_InitTypeDef gpio_init;

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
```

```
	gpio_init.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
	gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
	gpio_init.GPIO_OType = GPIO_OType_OD;
	GPIO_Init(GPIOB, &gpio_init);

	GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1);
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_I2C1);

	I2C_StructInit(&i2c_init);
	i2c_init.I2C_Mode = I2C_Mode_I2C;
	i2c_init.I2C_DutyCycle = I2C_DutyCycle_2;
	i2c_init.I2C_OwnAddress1 = 0x01;
	i2c_init.I2C_ClockSpeed = 1000;
	i2c_init.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
	i2c_init.I2C_Ack = I2C_Ack_Disable;
	I2C_Init(I2C1, &i2c_init);

	I2C_Cmd(I2C1, ENABLE);
}

void i2c_start(I2C_TypeDef * i2c, uint8_t address) {
	int temp;

	i2c->CR1 |= I2C_CR1_START;
	while(!(i2c->SR1 & I2C_SR1_SB));
	temp = i2c->SR1;

	i2c->DR = address;
	while(!(i2c->SR1 & I2C_SR1_ADDR));
	temp = i2c->SR1;
	temp = i2c->SR2;
}

void i2c_write(I2C_TypeDef * i2c, uint8_t data) {
	i2c->DR = data;
	while(!(i2c->SR1 & I2C_SR1_BTF));
}

uint8_t i2c_read(I2C_TypeDef * i2c) {
	while(!(i2c->SR1 & I2C_SR1_RXNE));
	return i2c->DR;
}

void i2c_stop(I2C_TypeDef * i2c) {
	i2c->CR1 |= I2C_CR1_STOP;
}
```

Как видно из программной реализации, все основные операции сопровождаются установкой битов в управляющем регистре 1 и проверкой регистра статуса путем опроса.

Проверку программной реализации проведем в режиме отладки (рис. 7). Значения инкремента цикла и считанных данных должны совпасть.

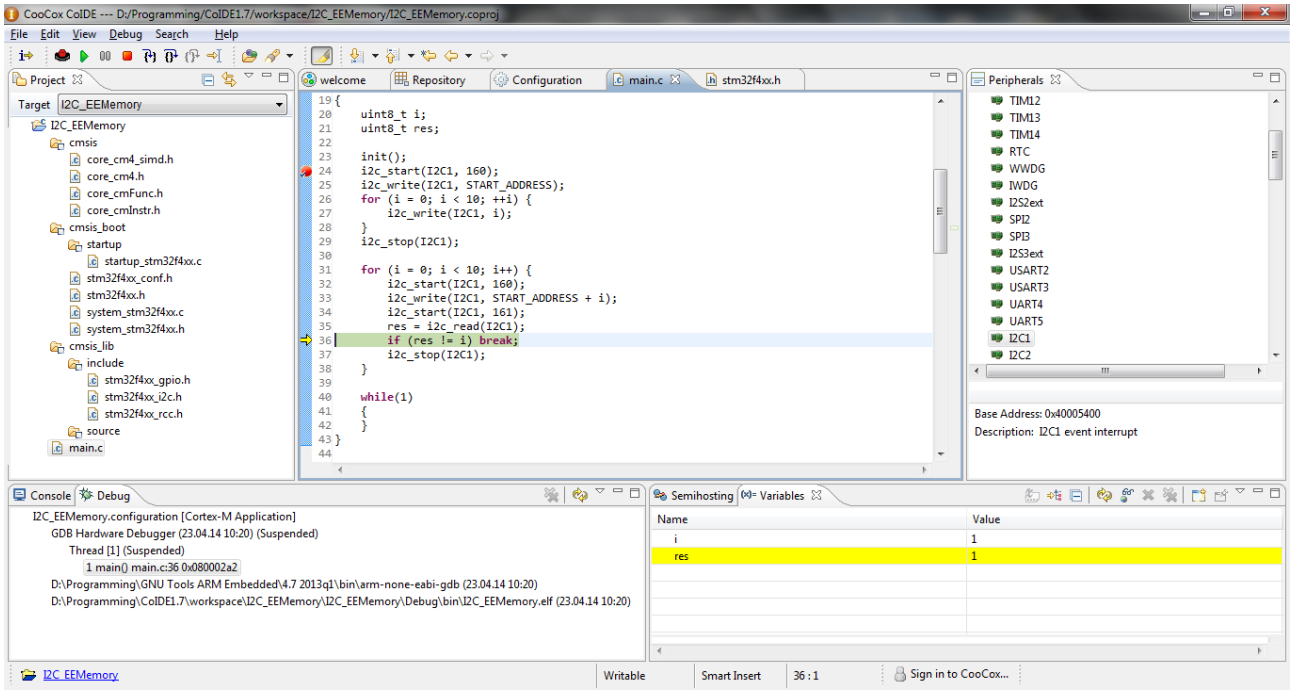


Рис. 7. Проверка считанных данных

Задание

1. Проверить работу программы, приведенной в примере. Попробовать изменить значения, которые передаются в память.
2. Ознакомится с документацией для микроконтроллера и использования I2C, и микросхемы памяти. Организовать считывание нескольких значений без остановки обмена (сигнал STOP). Важную роль в этом играет сигнал ACK, который передается со стороны микроконтроллера.
3. Продемонстрировать в режиме отладки значения, записанные и считанные в переменную-строку.

Лабораторная работа №3. Подключение матричной клавиатуры

Цель работы: исследовать подключение матричной клавиатуры и организацию выполнения действий с помощью опроса клавиатуры.

Оборудование и программное обеспечение: матричная клавиатура, отладочная плата STM32F3 Discovery.

Теоретические сведения

Клавиатура – едва ли не самый распространенный способ ввода информации в цифровых системах, который используется в компьютерах, мобильных телефонах и других устройствах.

Матричная клавиатура объединяет в своем составе кнопки, которые соединены в виде матрицы. Количество клавиш по горизонтали и вертикали может значительно отличаться в зависимости от предназначения. Для обеспечения защиты выводов рекомендуется использовать

токоограничительные резисторы при подключении матричной клавиатуры (для защиты в случае одновременного нажатия нескольких кнопок).

С точки зрения использования матричной клавиатуры при реализации устройств, то ее используют для обеспечения введения данных пользователем. С точки зрения программной реализации, наиболее распространенным методом подключения является организация опроса состояний клавиатуры с выводом соответствующих логических уровней на строки (столбцы). В случае, когда значение одного из разрядов в значении, которое считано с входов, отличается от остальных, то была нажата клавиша (рассмотрим пример нажатия только одной кнопки).

Пример программы

В данной работе рассмотрим подключение пленочной матричной клавиатуры (рис. 8), которая имеет 4 строки по 4 кнопки в каждой. В такой клавиатуре первые 4 контакта отвечают за подачу сигнала на строки, а последние 4 предназначены для считывания сигналов.

Рис. 8. Внешний вид пленочной матричной клавиатуры

Опрос клавиатуры организуем попеременной подачей высокого уровня сигнала на одну из строк, в то время как на остальные подается низкий уровень. Тогда определение нажатия клавиши будет проводится на основе сравнения считанного с входов значения (требует предварительной обработки) с 0. В случае появления сигнала высокого уровня на входе это значение будет отличаться от 0. Определить, какая именно клавиша была нажата, в данном случае можно проверив, какой именно разряд установлен в лог. 1.

Поскольку микроконтроллер на отладочном модуле имеет достаточно аппаратных ресурсов (несколько таймеров), то для реализации опроса клавиатуры используем два из них. В прерывании по переполнению первого выполняется переключение сигналов для строк. В обработчике второго выполняется считывание сигналов.

При реализации обработки нажатия клавиши на клавиатуре важно также определить, как именно обрабатывать удержание клавиши. Рассмотрим вариант, когда удержание приводит к выполнению определенного действия

только один раз, а все остальные значения игнорируются, пока кнопка не будет отпущена.

Нажатие клавиши на клавиатуре обычно приводит к выполнению определенных действий. В примере реализуем обработку клавиш с надписями 0, 1 и 2, которые будут включать соответствующее количество светодиодов, размещенных на отладочном модуле.

Программный код для реализации описанных действий представлен ниже.

```
#include "stm32f30x.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_rcc.h"
#include "stm32f30x_tim.h"

#define FIRST_OUT_PIN GPIO_Pin_1
#define SECOND_OUT_PIN GPIO_Pin_5
#define THIRD_OUT_PIN GPIO_Pin_7
#define FORTH_OUT_PIN GPIO_Pin_5

#define FIRST_OUT_PORT GPIOB
#define SECOND_OUT_PORT GPIOC
#define THIRD_OUT_PORT GPIOA
#define FORTH_OUT_PORT GPIOA

#define FIRST_OUT FIRST_OUT_PORT, FIRST_OUT_PIN
#define SECOND_OUT SECOND_OUT_PORT, SECOND_OUT_PIN
#define THIRD_OUT THIRD_OUT_PORT, THIRD_OUT_PIN
#define FORTH_OUT FORTH_OUT_PORT, FORTH_OUT_PIN

#define FIRST_IN_PIN GPIO_Pin_4
#define SECOND_IN_PIN GPIO_Pin_3
#define THIRD_IN_PIN GPIO_Pin_1
#define FORTH_IN_PIN GPIO_Pin_3

#define FIRST_IN_PORT GPIOF
#define SECOND_IN_PORT GPIOA
#define THIRD_IN_PORT GPIOA
#define FORTH_IN_PORT GPIOC

#define FIRST_IN FIRST_IN_PORT, FIRST_IN_PIN
#define SECOND_IN SECOND_IN_PORT, SECOND_IN_PIN
#define THIRD_IN THIRD_IN_PORT, THIRD_IN_PIN
#define FORTH_IN FORTH_IN_PORT, FORTH_IN_PIN

uint8_t out_state = 0;
uint8_t prev_line = 10, prev_col = 10;

void init(void);
void turn_on_line(uint8_t out);
uint8_t read_input_value(void);
void turn_on_1led(void);
void turn_on_2led(void);
```

```
void turn_off_leds(void);

void TIM2_IRQHandler(void) {
	TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
	switch (out_state) {
		case 0: out_state = 1;
			turn_on_line(1);
			break;
		case 1: out_state = 2;
			turn_on_line(2);
			break;
		case 2: out_state = 3;
			turn_on_line(3);
			break;
		case 3: out_state = 4;
			turn_on_line(4);
			break;
		case 4: out_state = 0;
			break;
	}
	turn_on_line(out_state);
}

void TIM3_IRQHandler(void) {
	uint8_t row_value;
	TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
	row_value = read_input_value();
	if (row_value > 0) {
		prev_col = row_value;
		prev_line = out_state;
		if (row_value == 1 && out_state == 1) {
			turn_on_1led();
		} else if (row_value == 2 && out_state == 1) {
			turn_on_2led();
		} else if (row_value == 2 && out_state == 4) {
			turn_off_leds();
		}
	} else {
		if (prev_line == out_state) {
			if (prev_col != 10) {
				prev_col = 10;
				prev_line = 10;
			}
		}
	}
}

int main(void) {
	init();
	__enable_irq();
	NVIC_EnableIRQ(TIM2_IRQn);
	NVIC_EnableIRQ(TIM3_IRQn);
	while(1) {
	}
}
```

```
void init(void) {
	GPIO_InitTypeDef gpio_init;
	TIM_TimeBaseInitTypeDef tim_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA |

	RCC_AHBPeriph_GPIOB |

	RCC_AHBPeriph_GPIOC |

	RCC_AHBPeriph_GPIOF |

	RCC_AHBPeriph_GPIOE, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Pin = FIRST_OUT_PIN;
	gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_Init(FIRST_OUT_PORT, &gpio_init);

	gpio_init.GPIO_Pin = SECOND_OUT_PIN;
	GPIO_Init(SECOND_OUT_PORT, &gpio_init);

	gpio_init.GPIO_Pin = THIRD_OUT_PIN;
	GPIO_Init(THIRD_OUT_PORT, &gpio_init);

	gpio_init.GPIO_Pin = FORTH_OUT_PIN;
	GPIO_Init(FORTH_OUT_PORT, &gpio_init);

	gpio_init.GPIO_Mode = GPIO_Mode_IN;
	gpio_init.GPIO_Pin = FIRST_IN_PIN;
	GPIO_Init(FIRST_IN_PORT, &gpio_init);

	gpio_init.GPIO_Pin = SECOND_IN_PIN;
	GPIO_Init(SECOND_IN_PORT, &gpio_init);

	gpio_init.GPIO_Pin = THIRD_IN_PIN;
	GPIO_Init(THIRD_IN_PORT, &gpio_init);

	gpio_init.GPIO_Pin = FORTH_IN_PIN;
	GPIO_Init(FORTH_IN_PORT, &gpio_init);

	gpio_init.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_13;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	GPIO_Init(GPIOE, &gpio_init);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 | RCC_APB1Periph_TIM3, ENABLE);
	TIM_TimeBaseStructInit(&tim_init);
	tim_init.TIM_Period = 10000;
	tim_init.TIM_Prescaler = 16;
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM2, &tim_init);
	TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

	tim_init.TIM_Period = 1000;
	TIM_TimeBaseInit(TIM3, &tim_init);
	TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);

	TIM_Cmd(TIM2, ENABLE);
	TIM_Cmd(TIM3, ENABLE);
}

void turn_on_line(uint8_t out){
	if (out == 1) {
```

```
            GPIO_WriteBit(FIRST_OUT, Bit_SET);
            GPIO_WriteBit(SECOND_OUT, Bit_RESET);
            GPIO_WriteBit(THIRD_OUT, Bit_RESET);
            GPIO_WriteBit(FORTH_OUT, Bit_RESET);
      } else if (out == 2) {
            GPIO_WriteBit(FIRST_OUT, Bit_RESET);
            GPIO_WriteBit(SECOND_OUT, Bit_SET);
            GPIO_WriteBit(THIRD_OUT, Bit_RESET);
            GPIO_WriteBit(FORTH_OUT, Bit_RESET);
      } else if (out == 3) {
            GPIO_WriteBit(FIRST_OUT, Bit_RESET);
            GPIO_WriteBit(SECOND_OUT, Bit_RESET);
            GPIO_WriteBit(THIRD_OUT, Bit_SET);
            GPIO_WriteBit(FORTH_OUT, Bit_RESET);
      } else if (out == 4) {
            GPIO_WriteBit(FIRST_OUT, Bit_RESET);
            GPIO_WriteBit(SECOND_OUT, Bit_RESET);
            GPIO_WriteBit(THIRD_OUT, Bit_RESET);
            GPIO_WriteBit(FORTH_OUT, Bit_SET);
      } else {
            GPIO_WriteBit(FIRST_OUT, Bit_RESET);
            GPIO_WriteBit(SECOND_OUT, Bit_RESET);
            GPIO_WriteBit(THIRD_OUT, Bit_RESET);
            GPIO_WriteBit(FORTH_OUT, Bit_RESET);
      }
}

uint8_t read_input_value(void) {
      return GPIO_ReadInputDataBit(FIRST_IN) +
                  (GPIO_ReadInputDataBit(SECOND_IN) << 1) +
                  (GPIO_ReadInputDataBit(THIRD_IN) << 2) +
                  (GPIO_ReadInputDataBit(FORTH_IN) << 3);
}

void turn_on_1led(void) {
      GPIO_WriteBit(GPIOE, GPIO_Pin_13, Bit_SET);
      GPIO_WriteBit(GPIOE, GPIO_Pin_14, Bit_RESET);
}

void turn_on_2led(void) {
      GPIO_WriteBit(GPIOE, GPIO_Pin_13, Bit_SET);
      GPIO_WriteBit(GPIOE, GPIO_Pin_14, Bit_SET);
}

void turn_off_leds(void) {
      GPIO_WriteBit(GPIOE, GPIO_Pin_13, Bit_RESET);
      GPIO_WriteBit(GPIOE, GPIO_Pin_14, Bit_RESET);
}
```

Как видно, для реализации отслеживания удержания и отпускания кнопки на клавиатуре используются две дополнительные переменные, которые обозначают координаты последней нажатой клавиши. Они инициализируются значением по умолчанию (10), а в обработчике прерываний таймера, в случае, если значение входов отличается от 0, им присваивается значение строки и столбца. В том случае, когда кнопка будет отпущена, им снова присваивается значение 10.

Задание

1. Проверить работу программы, приведенной в примере.
2. Организовать опрос всех кнопок на клавиатуре и сопоставить нажатию каждой кнопки отдельное действие.

Лабораторная работа №4. Использование датчиков. Работа с акселерометром

Цель работы: исследовать использование современных датчиков MEMS на примере акселерометра для использования в микропроцессорной технике.

Оборудование и программное обеспечение: отладочный модуль STM32F4 Discovery, среда разработки Keil uVision.

Теоретические сведения

Датчики – устройства, предназначенные для измерения определенных показателей. Чаще всего датчики используют ту особенность, что под действием внешних факторов (движение, изменение температуры, влияние химических веществ и др.) изменяется напряжение, что позволяет сопоставить эти изменения и получить информацию о том, какое именно внешнее влияние произошло, измерить его. Датчики имеют разную чувствительность – наименьшее значение изменения, которое может быть измерено, а также разный диапазон измеряемых значений. Датчики также не являются идеальными устройствами, поэтому для них характерно то, что даже при отсутствии внешнего воздействия, когда он должен выдавать значение 0, возможны незначительные изменения около непосредственно 0, что следует учитывать при построении систем с использованием датчиков. Примером может служить акселерометр, который в неподвижном состоянии может фиксировать незначительное ускорение, хотя движение отсутствует.

В последнее время большое распространение получили MEMS-датчики (Microelectromechanical systems). Они имеют небольшие размеры, но в то же время обеспечивают большой диапазон измеряемых значений и точность показателей. Именно такие датчики чаще всего используются в мобильной технике, робототехнике и др.

Акселерометры – датчики, которые позволяют измерять ускорение. Поскольку на основе данных ускорения можно получить данные про скорость, а также положение объекта, то эти датчики находят широкое применение, в том числе и в комбинации с другими датчиками (например, гироскопами – датчиками поворота). Важной характеристикой акселерометра является количество осей, относительно которых можно определить ускорение. Чаще всего, ускорение можно определить для 3 осей: X, Y, Z. Единицей измерения ускорения является м/с^2, однако, производители часто в документации указывают параметры с помощью величины ускорения свободного падения $g \approx 9.81$ *м/с*^{2}. Также возможно использование при описании единицы измерения gal (Galileo), что соответствует см/с^2.

Пример программы

В данной работе рассматривается использование акселерометра на примере устройства LIS302DL. Данное устройство уже присутствует на отладочной плате STM32F4 Discovery и подключено к выводам интерфейса SPI. Он способен измерять ускорение в диапазоне ±2g, ±8g в зависимости от настроек. Чувствительность также зависит от указанного диапазона. Акселерометр в неподвижном состоянии должен выдавать значения ускорения около 0 для осей X, Y, а для оси Z – 1 g. При повороте датчика составляющая свободного падения имеет влияние на горизонтальные оси. Для того, чтобы отключить такое влияние можно включить специальный фильтр, который обеспечит фильтрацию этой составляющей. При включенном фильтре значения ускорения для оси Z стает близким к 0, также, как и для остальных осей.

Считывание данных из акселерометра возможно с помощью интерфейса SPI или I2C. На плате он подключен к выводам модуля SPI. Обмен данными происходит в режиме 3 SPI: уровень сигнала по умолчанию для линии тактовой частоты – высокий, а считывание данных происходит при втором изменении сигнала на тактовой линии (из лог. 0 в лог. 1).

При работе с акселерометром следует помнить, что обновление данных происходит с определенной частотой, поэтому опрос данных следует проводить на основе сведений об этом. Например, если частота обновления информации – 100 Гц, то опрос с большей частотой может привести к тому, что будет фиксироваться движение, когда его нет.

В данное работе рассмотрим пример, который выполняет считывание данных из акселерометра и на основе полученной информации включает светодиоды на модуле в соответствии с направлением, в котором фиксируется движение. Частота считывания данных выбрана 100 Гц.

Рассмотрим детальнее процесс взаимодействия с акселерометром. В соответствии с документацией, акселерометр подключен к выводам модуля SPI1 в составе микроконтроллера, поэтому данные выводы должны быть сконфигурированы для работы в режиме альтернативной функции. Сам модуль SPI в данном случае выступает в качестве ведущего устройства. С процессом передачи/приема данных можно ознакомиться по рисунку 9.

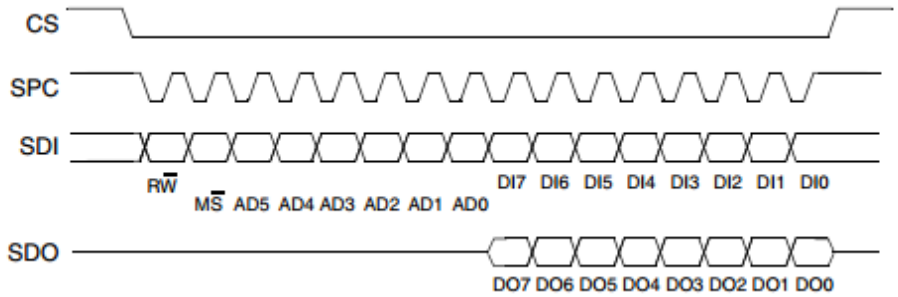


Рис. 9. Организация обмена данными между акселерометром и управляющим устройством

Начало обмена инициируется управляющим устройством и соответствует тому, что он отправляет акселерометру однобайтовую посылку. Первый бит в байте указывает направление, в котором производится обмен (0 – считывание, 1

– запись). Второй бит отвечает за то, инкрементируется ли адрес при следующей операции. Далее идут биты, которые обозначают адрес регистра, к которому производится обращение. 6 битов достаточно для того, чтобы обеспечить обращение к нужному количеству регистров. С адресами необходимых регистров можно ознакомится в документации. Наиболее важные регистры – регистры управления, регистры данных и регистры статуса.

В случае, когда необходимо выполнить запись, сразу за первым байтом управляющее устройство отсылает значение, которое необходимо записать. Если установлен бит инкремента регистра, то следующий отправленный байт запишется по адресу, больше текущего на 1. На линии передачи данных от акселерометра при этом данные отсутствуют.

В случае, когда необходимо выполнить считывание, можно отправить любое значение: акселерометр его проигнорирует. Отправка данных необходима только для того, чтобы на тактовой линии появился сигнал передачи. При этом происходит отправка данных, которые находятся по указанному адресу. Также есть возможность считывать несколько значений с инкрементом текущего адреса.

```
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_spi.h>
#include <stm32f4xx_tim.h>
#include <math.h>

#define DUMMY 0x00

uint8_t res, status;
int8_t reg_x, reg_y, reg_z;
double acc_x, acc_y;
double prev_speed_x = 0.0, current_speed_x;
double prev_speed_y = 0.0, current_speed_y;
double dist_x, dist_y, total_dist = 0.0;

void init(void);
void low_cs(void);
void high_cs(void);
void init_sensor(void);
void write_single_address(uint8_t address, uint8_t value);
uint8_t read_single_address(uint8_t address);
uint8_t write_byte(uint8_t byte);
void init_leds(void);

int main(void) {
	init();
	init_leds();
	init_sensor();
	while (1) {
		TIM2->CNT = 0;
		TIM_Cmd(TIM2, ENABLE);
		while(TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) == RESET);
		TIM_ClearFlag(TIM2, TIM_FLAG_Update);
		TIM_Cmd(TIM2, DISABLE);
		status = read_single_address(0x27);
		if ((status & 0x03 == 0x03)) {
			reg_x = read_single_address(0x29);
```

```
                    reg_y = read_single_address(0x2b);
                    GPIO_ResetBits(GPIOD,
                                GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15);
                    if (reg_x > 5 || reg_x < -5) {
                          if (reg_x > 0) {
                                GPIO_WriteBit(GPIOD, GPIO_Pin_15, Bit_SET);
                          } else if (reg_x < 0) {
                                GPIO_WriteBit(GPIOD, GPIO_Pin_13, Bit_SET);
                          }
                    }
                    if (reg_y > 5 || reg_y < -5) {
                          if (reg_y > 0) {
                                GPIO_WriteBit(GPIOD, GPIO_Pin_14, Bit_SET);
                          } else if (reg_y < 0) {
                                GPIO_WriteBit(GPIOD, GPIO_Pin_12, Bit_SET);
                          }
                    }
              }
        }
}

void init(void) {
        GPIO_InitTypeDef gpio_init;
        SPI_InitTypeDef spi_init;
        TIM_TimeBaseInitTypeDef tim_init;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

        GPIO_StructInit(&gpio_init);
        gpio_init.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
        gpio_init.GPIO_Mode = GPIO_Mode_AF;
        gpio_init.GPIO_OType = GPIO_OType_PP;
        gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
        gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &gpio_init);

        GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1); // SCK
        GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1); // MISO
        GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1); // MOSI

        gpio_init.GPIO_Mode = GPIO_Mode_OUT;
        gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
        gpio_init.GPIO_Pin = GPIO_Pin_3;
        GPIO_Init(GPIOE, &gpio_init);

        high_cs();

        SPI_StructInit(&spi_init);
        spi_init.SPI_Mode = SPI_Mode_Master;
        spi_init.SPI_CPHA = SPI_CPHA_2Edge;
        spi_init.SPI_CPOL = SPI_CPOL_High;
        spi_init.SPI_DataSize = SPI_DataSize_8b;
        spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
        spi_init.SPI_CRCPolynomial = 7;
        spi_init.SPI_NSS = SPI_NSS_Soft;
        spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
        spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
        SPI_Init(SPI1, &spi_init);

        TIM_TimeBaseStructInit(&tim_init);
```

```
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_Prescaler = 160 - 1;
	tim_init.TIM_Period = 1000 - 1;
	TIM_TimeBaseInit(TIM2, &tim_init);

	SPI_Cmd(SPI1, ENABLE);
}

void low_cs(void) {
	GPIO_WriteBit(GPIOE, GPIO_Pin_3, Bit_RESET);
}

void high_cs(void) {
	GPIO_WriteBit(GPIOE, GPIO_Pin_3, Bit_SET);
}

void init_sensor(void) {
	write_single_address(0x20, 0x47);
}

uint8_t read_single_address(uint8_t address) {
	uint8_t res;

	low_cs();
	write_byte(0x80 | address);
	res = write_byte(DUMMY);
	high_cs();
	return res;
}

void write_single_address(uint8_t address, uint8_t value) {
	low_cs();
	write_byte(address);
	write_byte(value);
	high_cs();
}

uint8_t write_byte(uint8_t byte) {
	while (SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE) == RESET)
		;
	SPI_SendData(SPI1, byte);
	while (SPI_GetFlagStatus(SPI1, SPI_FLAG_RXNE) == RESET)
		;
	return SPI_ReceiveData(SPI1);
}

void init_leds(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
	GPIO_Init(GPIOD, &gpio_init);
}
```

Для того, чтобы не происходило реагирование не допустимые отклонения от нуля, показатели, которые по модулю меньше определенного значения не учитываются. Используются данные для 2 осей X и Y. Движение в

определенном направлении приводит к тому, что светодиоды отображают, в каком направлении происходит движение.

Задание

1. Проверить работу программы, приведенной в примере.
2. Проверить значение ускорения свободного падения, которое возвращает датчик, на основе значения, полученного из регистра для оси Z. Продемонстрировать значение в обычном и перевернутом положении. Для этого использовать документацию датчика (см. Список источников).
3. На основе предложенного примера реализовать измерение перемещения по определенной оси. Учитывать чувствительность и диапазон измерений устройства в режиме передачи с частотой 100 Гц. Для получения перемещения дважды проинтегрировать по времени полученное значение.
4. Продемонстрировать значение переменной, которая описывает перемещение в режиме отладки.

СПИСОК ИСТОЧНИКОВ

1. STM32F3DISCOVERY Discovery kit for STM32F303xx microcontrollers [Электронный ресурс]. Режим доступа: URL: http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user_manual/DM00063382.pdf. – Подзагол. с экрана.
2. Discovery kit for STM32F407/417 lines [Электронный ресурс]. Режим доступа: URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf. – Подзагол. с экрана.
3. BR24C04-W [Электронный ресурс]. Режим доступа: URL: http://www.alldatasheet.com/datasheet-pdf/pdf/36313/ROHM/BR24C04-W.html. – Загол. с экрана.
4. LIS302DL [Электронный ресурс]. Режим доступа: URL: http://storm.cis.fordham.edu/~gweiss/wisdm-papers/Ipod-accelerometer.pdf. – Загол. с экрана.
5. I2C [Электронный ресурс]. Режим доступа: URL: http://uk.wikipedia.org/wiki/I2C. – Загол. с экрана.

ПРИЛОЖЕНИЕ А

Логический анализатор на базе модуля STM32F4 Discovery

Логический анализатор – устройство, предназначенное для отображения цифровых сигналов. Такой инструмент является незаменимым при изучении работы и проектировании цифровых схем, поскольку позволяет проверить, какие именно сигналы за определенный промежуток времени прошли через линии, которые подключены к входам логического анализатора. Поскольку, чаще всего, проверки требует не один, а сразу несколько каналов, то анализатор, обычно, имеет большое количество входов (например, 4, 8, 16).

На базе отладочного модуля также можно построить логический анализатор. Основой для него будет служить STM32F4 Discovery. Для преобразования платы в логический анализатор необходимы:

– программная реализация (доступна вместе с исходными кодами на сайте https://code.google.com/p/logicdiscovery/downloads/detail?name=LogicDiscoveryHabr.bin&can=2&q=);

– программное обеспечение-клиент логического анализатора (в примере рассмотрен Open Logic Sniffer – http://www.lxtreme.nl/ols/#Download/).

Логический анализатор на основе модуля должен распознаваться системой как виртуальный COM-порт, поэтому, возможно, понадобится также загрузить и установить драйвера (STM32 Virtual COM Port) для корректной работы устройства.

Более детальное описание – http://habrahabr.ru/post/165853/.