Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Отчёт
по лабораторной работе № 4
ПОСЛЕДОВАТЕЛЬНЫЙ ИНТЕРФЕЙС SPI. ЖКИ. АКСЕЛЕРОМЕТР
ВАРИАНТ 11

Выполнил: Студент группы 950503
Гуринович А. В.

Проверил: Ассистент кафедры ЭВМ
Богдан Е. В.

Минск 2022

1 Цели работы

В ходе выполнения лабораторной работы необходимо изучить принципы организации последовательного интерфейса SPI и подключения устройств на его основе на базе микроконтроллера MSP430F5529.

2 Исходные данные к работе

Для выполнения лабораторной работы используется плата MSP-EXP430F5529 с использованием среды разработки Code Composer Studio. В процессе выполнения работы требуется написать программу, которая получает измерения акселерометра по оси Z и отражает их на экране в левом нижнем углу с поворотом текста на -90 градусов. По нажатию кнопки S1 зеркально отражает результат по вертикали, используя команды для ЖКИ.

Снять временные диаграммы всех линий интерфейса SPI (USCI_B1).

Не допускается использовать иные заголовочные файлы, кроме msp430.h, а также использовать высокоуровневые библиотеки.

3 Теоретические сведения

3.1 Последовательный интерфейс SPI

Микроконтроллер MSP430F5529 содержит два устройства USCI (Universal Serial Communication Interface), каждый из которых имеет два канала. Первое из них, USCI_A поддерживает режимы UART (Universal Asynchronous Receiver/Transmitter), IrDA, SPI (Serial Peripheral Interface). Второе, USCI_B - режимы I2C (Inter-Integrated Circuit) и SPI.

Интерфейс SPI является синхронным дуплексным интерфейсом. Это значит, что данные могут передаваться одновременно в обоих направлениях и синхронизируются тактовым сигналом. Интерфейс поддерживает:

- обмен по 3 или 4 линиям;
- 7 или 8 бит данных;
- режим обмена: LSB (младший значащий бит) или MSB (старший значащий бит) первым;
- режим ведущий (Master) / ведомый (Slave);
- независимые для приема и передачи сдвиговые регистры;
- отдельные буферные регистры для приема и передачи;
- непрерывный режим передачи;
- выбор полярности синхросигнала и контроль фазы;
- программируемая частота синхросигнала в режиме Master;
- независимые прерывания на прием и передачу;
- операции режима Slave в LPM4.

Структура интерфейса SPI представлена на рисунке 3.1. Линии интерфейса:

- UCxSIMO — Slave In, Master Out (передача от ведущего к ведомому);

- UCxSOMI — Slave Out, Master In (прием ведущим от ведомого);
- UCxCLK — тактовый сигнал, выставляется Master-устройством;
- UCxSTE — Slave Transmit Enable. В 4-битном протоколе используется для нескольких Master устройств на одной шине. В 3-битном не используется.

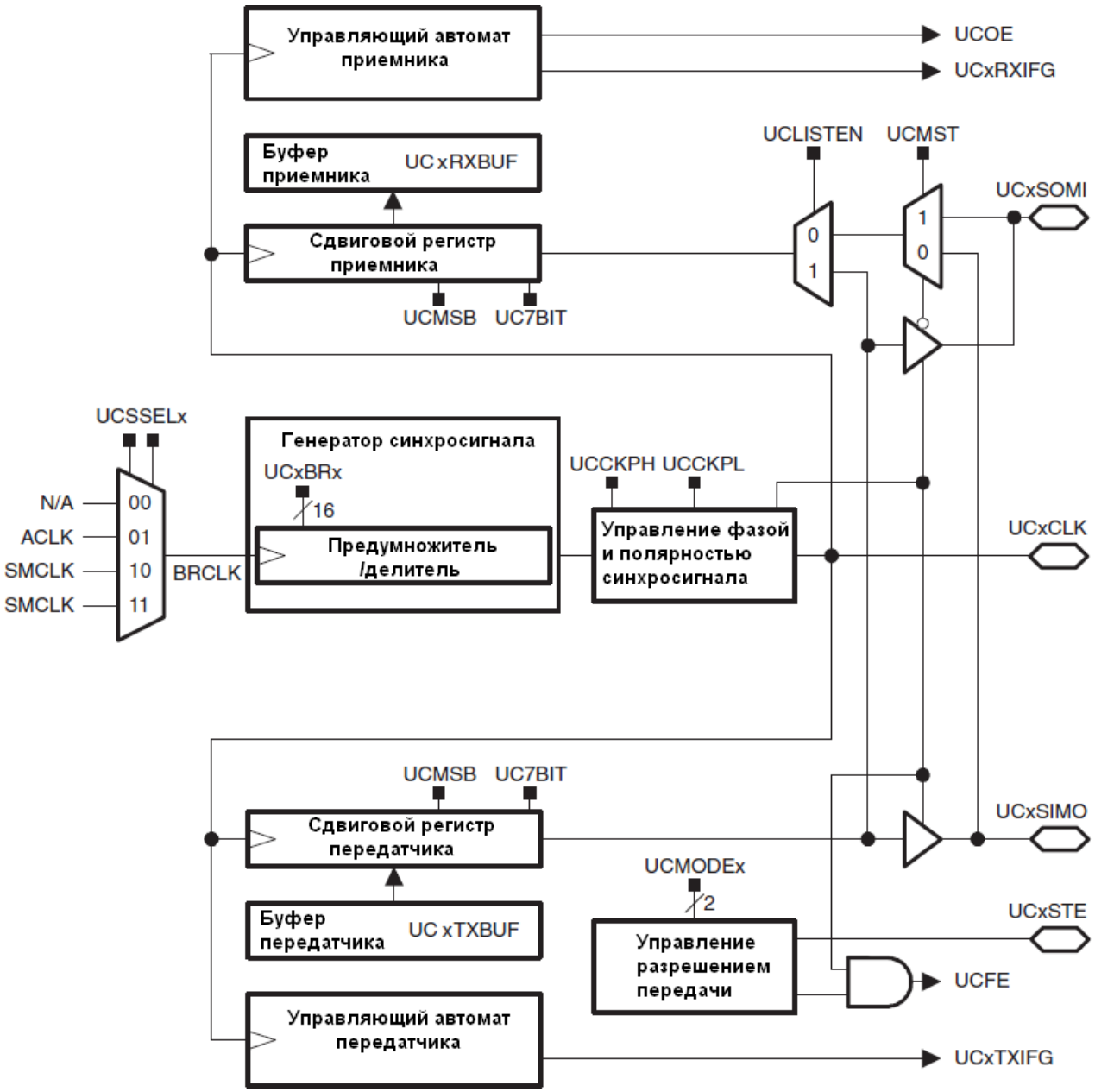

Рисунок 3.1 – Структура интерфейса SPI

Схема передачи данных начинает работу при помещении данных в буферный регистр передатчика UCxTXBUF. Данные автоматически помещаются в сдвиговый регистр (если он пуст), что начинает передачу по линии UCxSIMO. Флаг прерывания UCTXIFG устанавливается при перемещении данных в сдвиговый регистр и сигнализирует об освобождении буферного регистра, а не об окончания передачи. UCTXIFG требует локального и глобального разрешения прерываний UCTXIE и GIE,

автоматически сбрасывается при записи в буферный регистр передатчика UCxTXBUF.

Прием данных по линии UCxSOMI происходит автоматически и начинается с помещения данных в сдвиговый регистр приемника по спаду синхросигнала. Как только символ передан, данные из сдвигового регистра помещаются в буферный регистр приемника UCxRXBUF. После этого устанавливается флаг прерывания UCRXIFG, что сигнализирует об окончании приема. Аналогично, UCRXIFG требует локального и глобального разрешений прерываний UCRXIE и GIE, автоматически сбрасывается при чтении буферного регистра UCxRXBUF. Прием данных происходит только при наличии синхросигнала UCxCLK.

Сброс бита UCSWRST разрешает работу модуля USCI. Для Master-устройства тактовый генератор готов к работе, но начинает генерировать сигнал только при записи в регистр UCxTXBUF. Соответственно, без отправления данных (помещения в буферный регистр передатчика), тактовой частоты на шине не будет, и прием также будет невозможен. Для Slave-устройства тактовый генератор отключен, а передача начинается с выставлением тактового сигнала Master-устройством. Наличие передачи определяется флагом UCBUSY = 1.

Поля полярности UCCKPL и фазы UCCKPH определяют 4 режима синхронизации бит (см. рисунок 3.2).

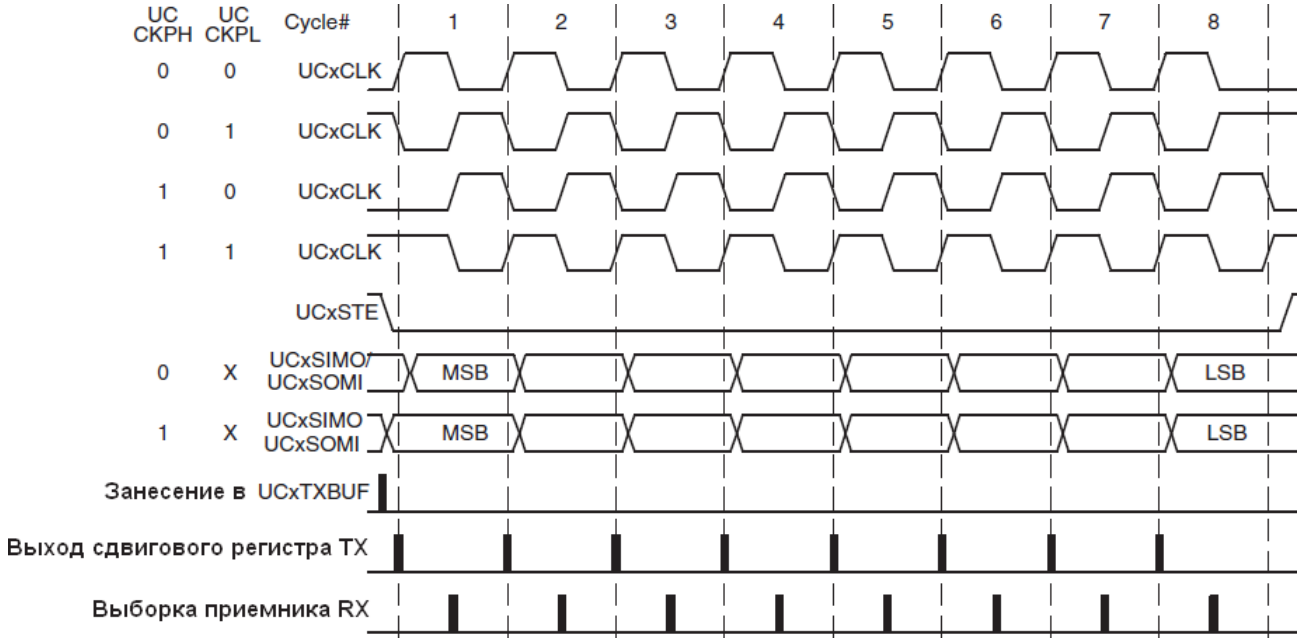

Рисунок 3.2 – Режимы синхронизации

Если UCMST = 1, для тактирования используется генератор USCI, источник входной частоты (ACLK или SMCLK) выбирается битами UCSSELx. 16 бит UCBRx (регистры UCxxBR1 и UCxxBR0) определяют делитель BRCLK входной тактовой частоты USCI: $f_{BitClock} = f_{BRCLK} / UCBRx$.

Состав и назначение регистров интерфейса SPI приведено в таблице 3.1,

а назначение полей — в таблице 3.2. Регистры всех каналов USCI в режиме SPI аналогичны, номер устройства (A или B) и номер канала (0 или 1) в именах указываются вместо xx, например, UCA0CTL0. Адреса регистров каналов USCI_B0 – 05E0h – 05FEh, USCI_A1 – 0600h – 061Eh, USCI_B1 – 0620h – 063Eh. После сброса поля всех регистров устанавливаются в 0, за исключением полей UCSWRST, UCTXIFG, которые устанавливаются в 1 (сброс и флаг готового буфера передатчика соответственно), и полей UCBRx, UCRXBUFx, UCTXBUFx, состояние которых не определено. Соответственно устанавливается 3-pin режим, ведомый (Slave), 8 бит данных, LSB, активный высокий уровень синхросигнала, по фронту синхросигнала данные выставляются на шину, по спаду — читаются (захватываются).

Таблица 3.1 – Регистры интерфейса SPI

Регистр	Адрес канала A0	Назначение
UCxxCTL0	05C1h	Регистры управления
UCxxCTL1	05C0h	
UCxxBR0	0506h	Управление скоростью передачи
UCxxBR1	0507h	
UCxxSTAT	050Ah	Регистр состояния
UCxxRXBUF	050Ch	Буфер приемника
UCxxTXBUF	050Eh	Буфер передатчика
UCxxIE	05DCh	Разрешение прерываний
UCxxIFG	05DDh	Флаги прерываний
UCxxIV	05DEh	Вектор прерываний

Таблица 3.2 – Поля регистров интерфейса SPI

Регистр	Биты	Поле	Назначение	Определение флагов в msp430f5529.h
1	2	3	4	5
UCAxCTL0	7	UCCKPH	Фаза Ти (0 — изменение по первому перепаду, захват по второму, 1 — наоборот)	UCCKPH
	6	UCCKPL	Полярность Ти (0 — активный - высокий)	UCCKPL
	5	UCMSB	Порядок передачи: 0 — LSB, 1- MSB	UCMSB
	4	UC7BIT	Разрядность: 0 — 8 бит, 1 — 7	UC7BIT
	3	UCMST	Режим: 0 — Slave, 1 – Master	UCMST

Продолжение таблицы 3.2.

1	2	3	4	5
UCAxCCTL0	1-2	UCMODEx	Синхронный режим: 00 – 3pin SPI, 01 – 4pin SPI + STE активный высокий, 10 – 4pin SPI + STE активный низкий, 11 – I^2C	UCMODE_0 … UCMODE_3
	0	UCSYNC	Режим: синхронный - 1	UCSYNC
UCAxCTL1	6-7	UCSSELx	Выбор источника Ти: 01 — ACLK, 10,11 - SMCLK	UCSSEL0, UCSSEL1
	0	UCSWRST	Разрешение программного сброса: 1 — логика интерфейса переводится в состояние сброса	UCSWRST
UCAxBR0	0-7	UCBRx	Младший байт делителя частоты	UCA0BR0
UCAxBR1	0-7	UCBRx	Старший байт делителя частоты	UCA0BR1
UCAxSTAT	7	UCLISTEN	Режим прослушивания — передача передается на прием	UCLISTEN
	6	UCFE	Флаг ошибки фрейма. При конфликте нескольких устройств на шине 4-pin	UCFE
	5	UCOE	Флаг ошибки перезаписи. Устанавливается, если происходит запись в регистр UCxRXBUF до чтения предыдущего значения	UCOE
	0	UCBUSY	Флаг приема/передачи	UCBUSY
UCAxRXBUF	0-7	UCRXBUFx	Буфер приемника	UCA0RXBUF
UCAxTXBUF	0-7	UCTXBUFx	Буфер передатчика	UCA0TXBUF
UCAxIE	1	UCTXIE	Разрешение прерывания передатчика	UCTXIE
	0	UCRXIE	Разрешение прерывания приемника	UCRXIE
UCAxIFG	1	UCTXIFG	Флаг прерывания передатчика	UCTXIFG
	0	UCRXIFG	Флаг прерывания приемника	UCRXIFG
UCAxIV	0-15	UCIVx	Вектор прерываний	UCA0IV

Все поля регистров UCxxCTL0, UCxxBRx, а также поле UCSSELx регистров UCxxCTL1 и поле UCLISTEN регистров UCxxSTAT могут быть изменены только при UCSWRST = 1.

На экспериментальной плате MSP-EXP430F5529 к устройству USCI_B, канал 1, в режиме SPI подключен ЖКИ экран EA DOGS102W-6 разрешением 102 x 64 пикселя, а к устройству USCI_A, канал 0, в режиме SPI подключен 3-осевой акселерометр CMA3000-D01.

3.2 ЖКИ экран DOGS102W-6

ЖКИ экран DOGS102W-6 поддерживает разрешение 102 x 64 пикселя, с подсветкой EA LED39x41-W, и управляется внутренним контроллером UC1701. Ток потребления составляет 250 мкА, а частота тактирования до 33 МГц при 3,3 В. Контроллер поддерживает 2 параллельных 8-битных режима и последвательный режим SPI, поддерживает чтение данных (в SPI режиме только запись). Устройство содержит двухпортовую статическую DDRAM.

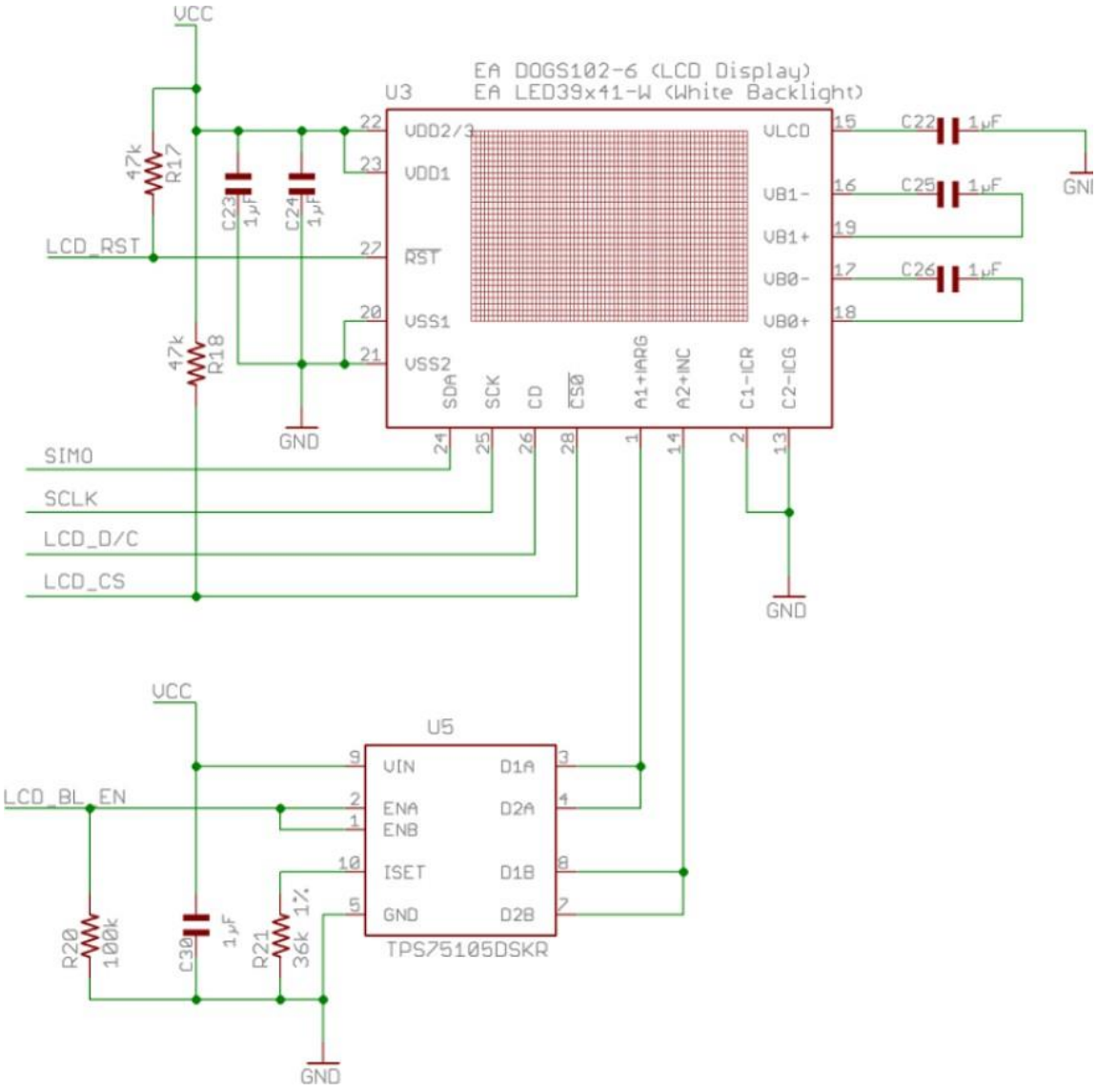

Рисунок 3.3 – Схема подключения ЖКИ экрана

Схема подключения экрана приведена на рисунке 3.3, соответствие выводов устройства выводам микроконтроллера MSP430F5529 и их назначение приведены в таблице 3.3

Таблица 3.3 – Соответствие выводов ЖКИ экрана

Выводы DOGS102W-6	Обозначение линии на схеме	Назначение	Вывод MSP430F5529	Требуемый режим
RST	LCD_RST	Сброс (= 0)	P5.7/TB0.1	P5.7
SDA	SIMO	SIMO данные	P4.1/ PM_UCB1SIMO/ PM_UCB1SDA	PM_UCB1SIMO
SCK	SCLK	Синхросигнал	P4.3/ PM_UCB1CLK/ PM_UCA1STE	PM_UCB1CLK
CD	LCD_D/C	Режим: 0 — команда, 1 — данные	P5.6/TB0.0	P5.6
CS0	LCD_CS	Выбор устройства (= 0)	P7.4/TB0.2	P7.4
ENA, ENB	LCD_BL_EN	Питание подсветки	P7.6/TB0.4	P7.6

Поскольку выбор устройства подключен к цифровому выходу, то управлять сигналом выбора устройства придется программно, фактически используется только 2 линии USCI микроконтроллера MSP430F5529 в режиме SPI.

Временные диаграммы обмена с устройством приведены на рисунке 3.4. ЖКИ поддерживает только запись, формат передачи MSB, чтение данных по фронту синхросигнала, Slave. Сигнал CD определяет, что передается в текущем байте — команда или данные, он считывается при передаче последнего бита.

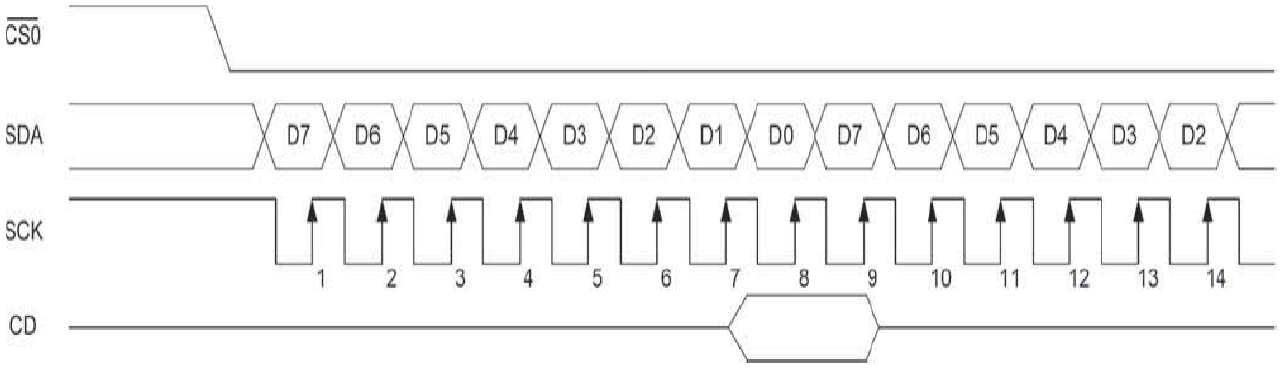

Рисунок 3.4 Временная диаграмма обмена с ЖКИ

Формат команд ЖКИ представлен в таблице 3.4.

Таблица 3.4 – Команды контроллера ЖКИ

Вход CD	Код команды, побитно								Описание
	7	6	5	4	3	2	1	0	
1	Биты данных D[7..0]								Запись одного байта данных в память
0	0	0	0	0	CA[3..0]				Установка номера столбца CA=0..131. Двухбайтная команда, младший полубайт передается первым байтом команды, старший полубайт — вторым. После сброса = 0
	0	0	0	1	CA[7..4]				
	0	0	1	0	1	PC[2..0]			Управление питанием. PC[0] – усилитель, PC[1] — регулятор, PC[2] — повторитель. 0 — отключено, 1 — включено. После сброса = 0
	0	1	SL[5..0]						Установка начальной линии скроллинга SL=0..63. После сброса = 0 (без скроллинга)
	1	0	1	1	PA[3..0]				Установка номера страницы PA=0..7. После сброса = 0
	0	0	1	0	0	PC[5..3]			Установка уровня внутреннего резисторного делителя PC=[0..7]. Используется для управления контрастом. После сброса = 100
	1	0	0	0	0	0	0	1	Регулировка контраста. Двухбайтная команда. PM=0..63. После сброса = 100000
	0	0	PM[5..0]						
	1	0	1	0	0	1	0	C1	Включение всех пикселей. 0 – отображение содержимого памяти, 1 – все пиксели включены (содержимое памяти сохраняется). После сброса = 0
	1	0	1	0	0	1	1	C0	Включение инверсного режима. 0 — нормальное отображение содержимого памяти, 1 — инверсное. После сброса = 0
	1	0	1	0	1	1	1	C2	Отключение экрана. 0 — экран отключен, 1 — включен. После сброса = 0
	1	0	1	0	0	0	0	MX	Порядок столбцов при записи в память 0 — нормальный (SEG 0-131), 1 — зеркальный (SEG 131-0). После сброса = 0
	1	1	0	0	MY	0	0	0	Порядок вывода строк 0 — нормальный (COM 0-63), 1 — зеркальный (COM 63-0). После сброса = 0
	1	1	1	0	0	0	1	0	Системный сброс. Данные в памяти не изменяются
	1	0	1	0	0	0	1	BR	Смещение напряжения делителя: 0 – 1/9, 1 – 1/7. После сброса = 0
	1	1	1	1	1	0	1	0	Расширенное управление. TC — температурная компенсация 0 = -0.05, 1=-0.11%/°C; WC – циклический сдвиг столбцов 0 = нет, 1 = есть; WP – циклический сдвиг страниц 0 = нет, 1 = есть. После сброса TC = 1, WC = 0, WP = 0
	TC	0	0	1	0	0	WC	WP	

Поля PC[2..0], C1, C0, C2, MX, BR при программном сбросе не устанавливаются. Поскольку контроллер поддерживает больше столбцов (132), чем у экрана (102), то можно задать пиксель за его границами. По этой же причине в зеркальном режиме номера столбцов соответствуют диапазону 30 — 131. Зеркальный режим столбцов (бит MX) не оказывает влияния на порядок вывода столбцов, поэтому данные, уже имеющиеся в памяти, будут отображаться одинаково в обоих режимах. При зеркальном режиме изменяется адрес записи байта в память. Подробнее режимы ориентации экрана (и вывода строк и столбцов) изображены на рисунке 3.5.Так, например, в режиме MX=0, MY=0, SL=0 (Прямой вывод без скроллинга), чтобы получить изображение, приведенное на рисунке, в столбец 1 страницу 0 должно быть записано значение 11100000b, а в столбец 2 страницу 0 — значение 00110011b.

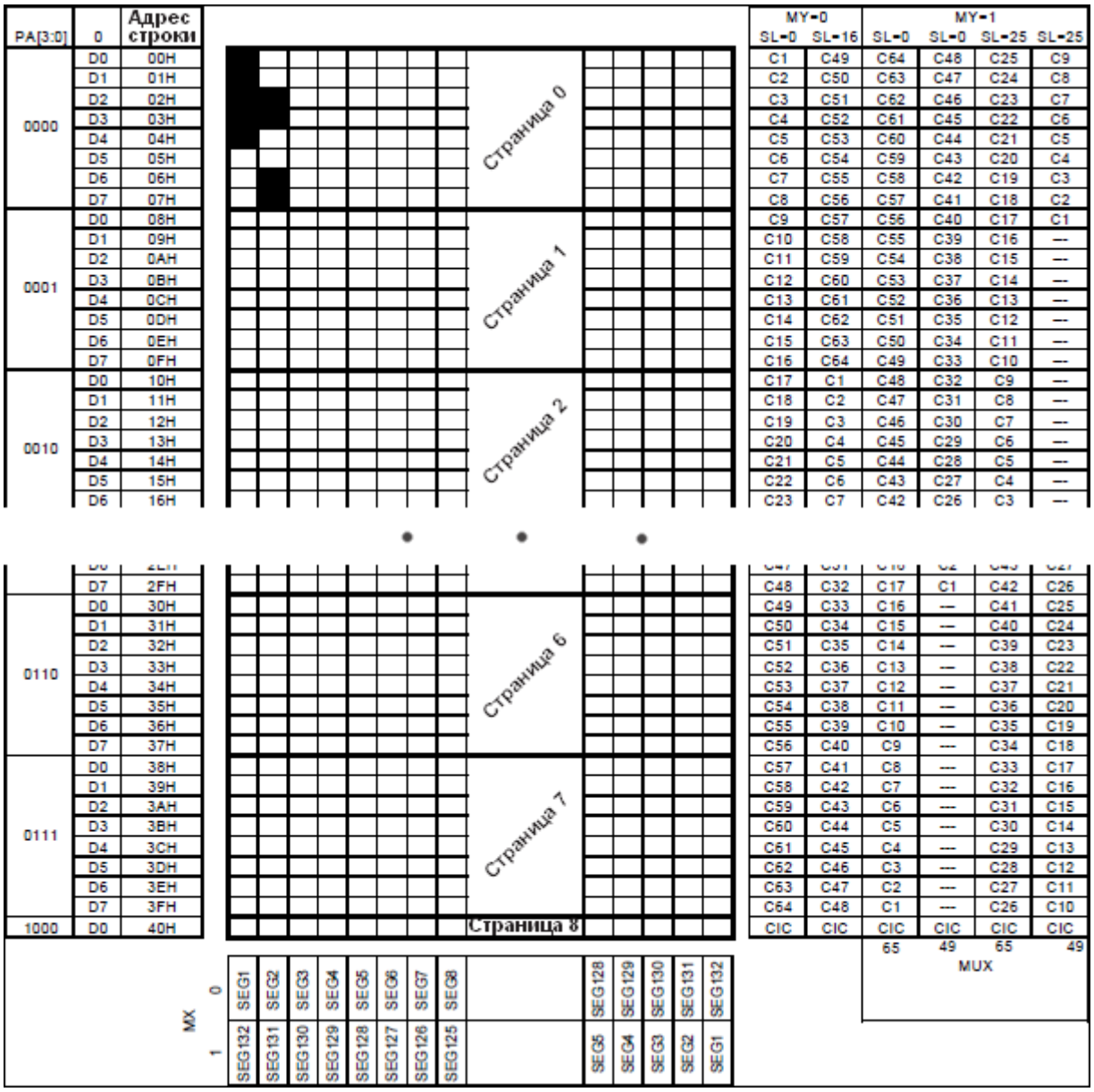

Рисунок 3.5 – Режимы ориентации экрана и вывода строк и столбцов

Для того, чтобы занесенное в память изображение при перевороте экрана «вверх ногами» выглядело точно так же, следует сместить нумерацию колонок на 30 позиций (при этом режим на зеркальный не меняется), а вывод строк изменить на зеркальный (см. рисунок 3.6).

Рисунок 3.6 – Ориентация экрана

Типичная последовательность инициализации выглядит следующим образом:

- 0x40 — установка начальной строки скроллинга =0 (без скроллинга);
- 0xA1 — зеркальный режим адресации столбцов;
- 0xC0 – нормальный режим адресации строк;
- 0xA4 – запрет режима включения всех пикселей (на экран отображается содержимое памяти);
- 0xA6 – отключение инверсного режима экрана;
- 0xA2 – смещение напряжения делителя 1/9;
- 0x2F – включение питания усилителя, регулятора и повторителя;
- 0x27, 0x81, 0x10 – установка контраста;
- 0xFA, 0x90 – установка температурной компенсации -0.11%/°C;
- 0xAF – включение экрана.

Типичная последовательность действий при включении питания, входе и выходе в режим ожидания и при выключении питания изображены на рисунке 3.7. Контроллер ЖКИ при формировании сигнала сброса требует ожидания 5-10 мс, при включении питания ожидания не требуется.

Подробно о командах и работе с устройством можно прочитать в документации [23, 24].

Для работы с устройством на программном уровне вначале необходимо установить требуемый режим соответствующих выводов микроконтроллера, далее задать режим работы интерфейса USCI. После этого можно передавать команды на ЖКИ с учетом того, что уровень сигнала на части линий необходимо задавать вручную.

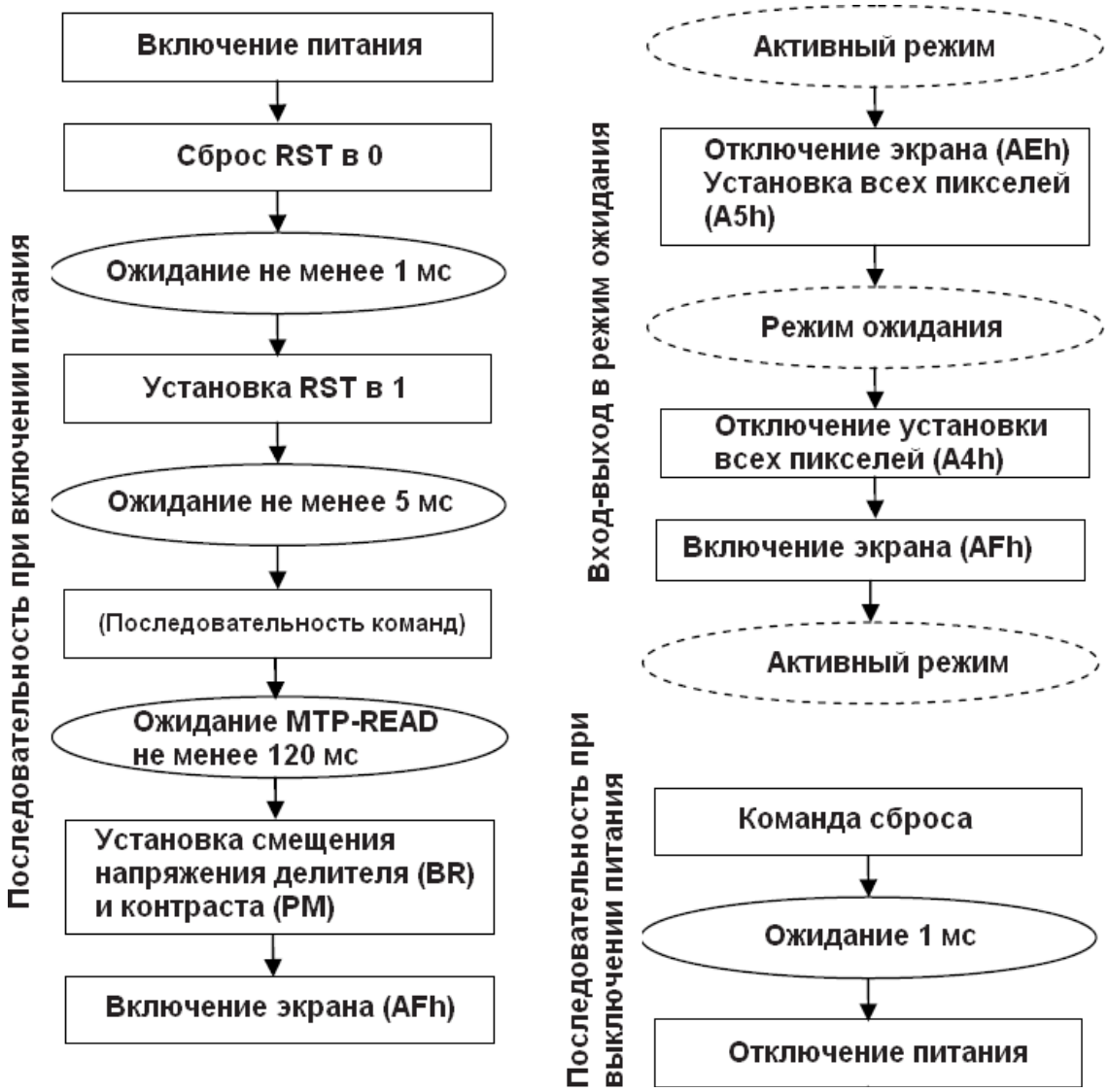

Рисунок 3.7 – Последовательность действий при включении/выключении ЖКИ и при входе/выходе в режим ожидания

3.3 Акселерометр CMA3000-D01

3-координатный акселерометр с цифровым выходом CMA3000-D01 обладает следующими возможностями:

- диапазон измерений задается программно (2g, 8g);
- питание 1.7 — 3.6 В;
- интерфейс SPI или I^2C задается программно;
- частота отсчетов (10, 40, 100, 400 Гц) задается программно;
- ток потребления в режиме сна 3 мкА;
- ток потребления при 10 отсчетах/сек — 7 мкА, при 400 отсчетах/сек — 70 мкА;
- максимальная тактовая частота синхросигнала 500 КГц;
- разрешение 18 mg (при диапазоне 2g), 71mg (при диапазоне 8g);
- чувствительность 56 точек / g (при 2g), 14 точек / g (при 8g);

- режимы обнаружения движения и обнаружения свободного падения.

Схема подключения акселерометра на макете MSP-EXP430F5529 приведена на рисунке 3.8, соответствие выводов устройства выводам микроконтроллера MSP430F5529 и их назначение приведены в таблице 3.5.

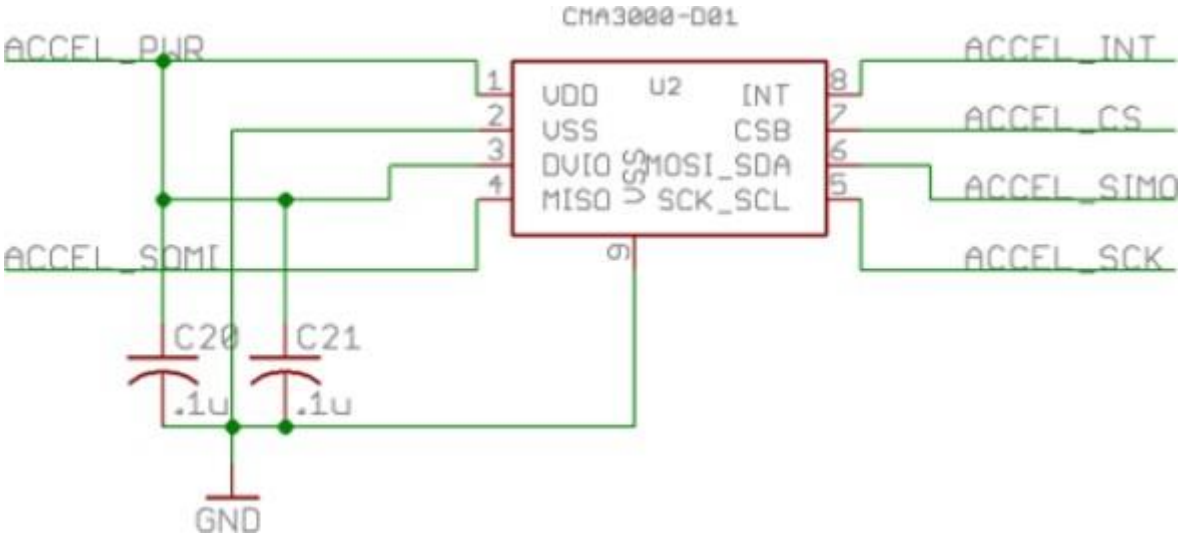

Рисунок 3.8 – Схема подключения акселерометра

Таблица 3.5. Соответствие выводов акселерометра

Выводы CMA3000-D01	Обозначение линии на схеме	Назначение	Вывод MSP430F5529	Требуемый режим
VDD, DVIO	ACCEL_PWR	Напряжение питания	P3.6/TB0.6	P3.6
MISO	ACCEL_SOMI	Линия приема данных по интерфейсу SPI	P3.4 / UCA0RXD / UCA0SOMI	UCA0SOMI
INT	ACCEL_INT	Сигнал прерывания	P2.5/TA2.2	P2.5
CSB	ACCEL_CS	Выбор устройства	P3.5/TB0.5	P3.5
MOSI_SDA	ACCEL_SIMO	Линия передачи данных по интерфейсу SPI	P3.3 / UCA0TXD / UCA0SIMO	UCA0SIMO
SCK_SCL	ACCEL_SCK	Синхросигнал	P2.7 / UCB0STC / UCA0CLK	UCA0CLK

В стандартном режиме измерения акселерометр работает со следующими сочетаниями диапазона измерений и частоты отсчетов: 2g — 400 Гц, 100 Гц; 8g — 400 Гц, 100 Гц, 40 Гц. В этом режиме используется фильтрация нижних частот, прерывание выставляется при готовности новых данных и может быть отключено программно. Флаг прерывания сбрасывается автоматически при чтении данных.

В режиме определения свободного падения допустимы следующие сочетания диапазона измерений и частоты отсчетов: 2g — 400 Гц, 100 Гц; 8g — 400 Гц, 100 Гц. Аналогично используется фильтр нижних частот,

прерывание выставляется при обнаружении свободного падения, при этом пороги срабатывания (время, ускорение) могут изменяться программно.

Режим определения движения использует только диапазон 8g с частотой отсчетов 10 Гц. В этом режиме происходит фильтрация по полосе пропускания 1,3 — 3,8 Гц, а прерывание выставляется при обнаружении движения. Пороги срабатывания (время, ускорение) могут изменяться программно, кроме того, может быть установлен режим перехода в режим измерения 400 Гц после обнаружения движения.

Сигнал сброса формируется внутренней цепью. После сброса читаются калибровочные и конфигурационные данные, хранящиеся в памяти. Бит PERR=0 регистра STATUS определяет успешность чтения этих данных. Запись последовательности 02h, 0Ah, 04h в RSTR регистр выполняет программный сброс устройства. После инициализации по сбросу акселерометр автоматически переходит в режим отключенного питания. Состояние регистров данных в этом режиме сохраняется. Программно этот режим устанавливается битами MODE = 000b или 111b в CTRL регистре.

Состав и назначение регистров и отдельных полей регистров акселерометра приведены в таблицах 3.6 — 3.7.

Таблица 3.6 – Регистры акселерометра

Регистр	Адрес	Чтение/ запись	Назначение
WHO_AM_I	0h	R	Идентификационный регистр
REVID	1h	R	Версия ASIC
CTRL	2h	RW	Регистр управления
STATUS	3h	R	Регистр состояния
RSTR	4h	RW	Регистр сброса
INT_STATUS	5h	R	Регистр состояния прерывания
DOUTX	6h	R	Регистр данных канала X
DOUTY	7h	R	Регистр данных канала Y
DOUTZ	8h	R	Регистр данных канала Z
MDTHR	9h	RW	Регистр порога ускорения в режиме обнаружения движения
MDFFTMR	Ah	RW	Регистр порога времени в режимах обнаружения движения и свободного падения
FF_THR	Bh	RW	Регистр порога ускорения в режиме обнаружения свободного падения
I2C_ADDR	Ch	R	Адрес устройства для протокола I^2C

Выбор интерфейса (SPI или I^2C) осуществляется при помощи сигнала выбора кристалла, при этом I^2C может быть отключен программно. Акселерометр всегда работает в ведомом (Slave) режиме по 4-проводному

соединению. Физические эквиваленты измеренного значения каждого бита в зависимости от режима приведены на рисунке 3.9.

Таблица 3.7 – Отдельные поля регистров акселерометра

Регистр	Биты	Поле	Назначение
CTRL	7	G_RANGE	Диапазон. 0 — 8g, 1 - 2g
	6	INT_LEVEL	Активный уровень сигнала прерывания: 0 - высокий, 1 - низкий
	5	MDET_EXIT	Переход в режим измерения после обнаружения движения
	4	I2C_DIS	Выбор интерфейса I^2C: 0 — разрешен, 1 - запрещен
	1-3	MODE[2..0]	Режим: 000 — отключено питание 001 — измерение, 100 Гц 10 — измерение, 400 Гц 11 — измерение, 40 Гц 100 — обнаружение движения, 10 Гц 101 — обнаружение свободного падения, 100 Гц 110 — обнаружение свободного падения, 400 Гц 111 — отключено питание
	0	INT_DIS	Запрещение прерывания (1 - отключен)
STATUS	3	PORST	Флаг состояния сброса. Чтение всегда сбрасывает в 0
	0	PERR	Флаг ошибки четности EEPROM
RSTR	0-7	RSTR	Запись 02h, 0Ah, 04h выполняет сброс устройства
INT_STATUS	2	FFDET	Флаг обнаружения свободного падения
	0-1	MDET[1..0]	Флаг обнаружения движения: 00 — нет, 01 - X , 10 - Y, 11 - Z

Диапазон	G_RANGE	Частота отсчетов	B7	B6	B5	B4	B3	B2	B1	B0
2g	1	400 Hz, 100 Hz	s	1142	571	286	143	71	36	1/56 = 18 mg
2g	1	40 Hz, 10 Hz	s	4571	2286	1142	571	286	143	1/14 = 71 mg
8g	0	400 Hz, 100 Hz	s	4571	2286	1142	571	286	143	1/14 = 71 mg
8g	0	40 Hz, 10 Hz	s	4571	2286	1142	571	286	143	1/14 = 71 mg

s = знак

Рисунок 3.9 – Физический эквивалент отдельных бит при измерении

Формат фрейма для одного обмена с устройством приведен на рисунке 3.10. Фрейм содержит 2 байта (16 бит). Первый байт содержит адрес регистра (первые 6 бит) и тип операции (R/W, 7 бит), 8 бит = 0. Второй байт содержит данные (при записи), и что угодно (при чтении). Поскольку тактовый сигнал выставляется на линию Master-устройством, то при чтении все-равно

необходимо выполнять холостую операцию записи. Данные заносятся в регистр по переднему фронту синхросигнала. При этом на линии MISO в первом байте первый бит не определен, второй — 0, потом 3 бита статуса сброса, далее следует 010, а второй байт при операции чтения содержит данные. При высоком CSB (устройство не выбрано), линия MISO находится в высокоимпедансном состоянии. Данные выставляются на MISO по заднему фронту, поэтому читать линию надо по переднему фронту. Пример операции чтения приведен на рисунке 3.11, а допустимые временные задержки — на рисунке 3.12.

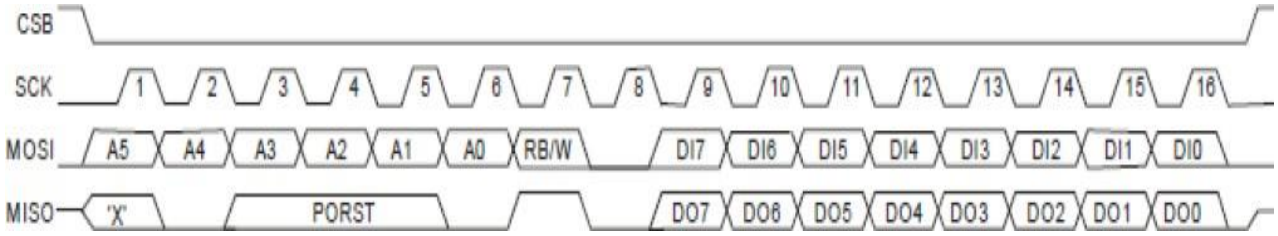

Рисунок 3.10 – Формат фрейма

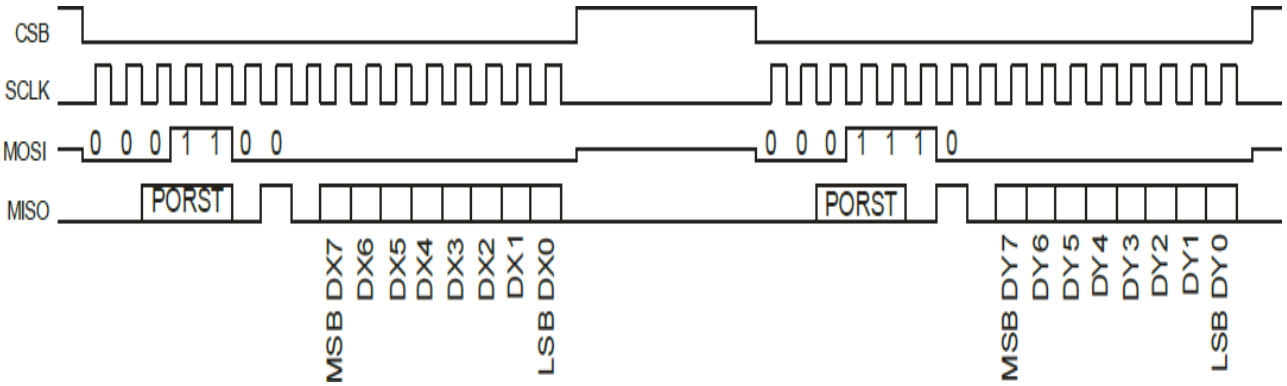

Рисунок 3.11 – Пример операции чтения данных

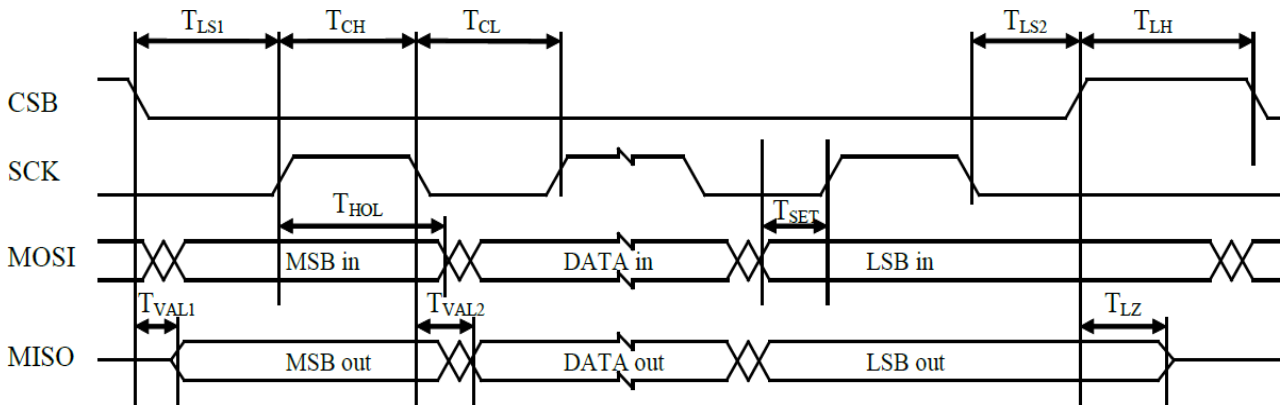

Рисунок 3.12 – Временные параметры обмена

На рисунке обозначены следующие временные соотношения, которые необходимы для нормального функционирования акселерометра:

- T_{LS1} — время от CSB до SCK, не менее 0,8 мкс;
- T_{LS2} — время от SCK до CSB, не менее 0,8 мкс;
- T_{CL} — длительность низкого SCK, не менее 0,8 мкс;
- T_{CH} — длительность высокого SCK, не менее 0,8 мкс;
- T_{SET} — время установки данных (до SCK), не менее 0,5 мкс;
- T_{HOL} — время удержания данных (от SCK до изменения MOSI), не менее 0,5 мкс;
- T_{VAL1} — время от CSB до стабилизации MISO, не более 0,5 мкс;
- T_{LZ} — время от снятия CSB до высокоимпедансного MISO, не более 0,5 мкс;
- T_{VAL2} — время от спада SCK до стабилизации MISO, не более 0,75 мкс;
- T_{LH} — задержка между SPI циклами (высокий CSB), не менее 22 мкс.

На рисунке 3.13 приведена типичная последовательность действий при инициализации акселерометра.

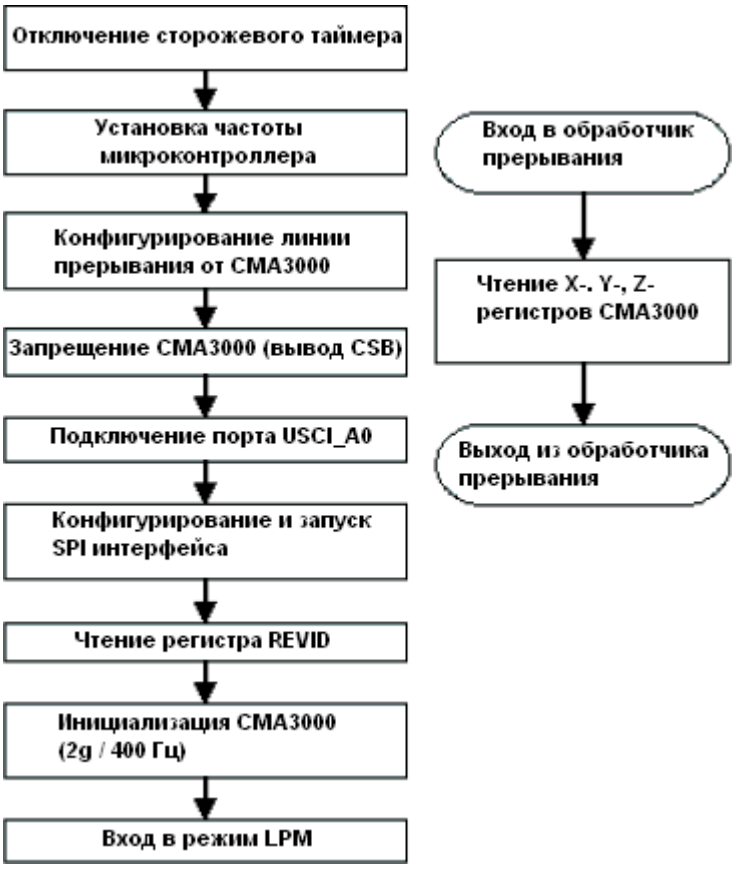

Рисунок 3.13 – Типичная последовательность при инициализации CMA3000-D01

Подробно о командах и работе с устройством можно прочитать в документации [25, 26].

Для работы с устройством на программном уровне вначале необходимо установить требуемый режим соответствующих выводов микроконтроллера,

далее задать режим работы интерфейса USCI. После этого можно передавать команды на акселерометр с учетом того, что уровень сигнала на части линий необходимо задавать вручную.

3.4 Измерение

Линии интерфейса SPI (USCI_B1) микроконтроллера выведены на разъем J5, и их можно наблюдать с помощью внешних приборов, например, осциллографа либо мультиметра. По каналу B1 подключен ЖКИ экран. Состав этих выводов указан в таблице 3.8. Уровень сигнала на линии можно измерить, подключив щупы осциллографа к соответствующему выводу разъема и GND разъема J5.

Таблица 3.8 – Выводы разъема J5

Vcc	P7.0, CB8, A12
P4.2, UCB1SOMI, UCB1SCL - SD	P7.1, CB9, A13
P4.1, UCB1SIMO, UCB1SDA – LCD/SD	P7.2, CB10, A14
P4.3, UCB1CLK, UCA1STE – LCD/SD	P7.3, CB11, A15
P4.0, UCB1STE, UCA1CLK – RF	P4.1, UCB1SIMO, UCB1SDA – LCD/SD
P3.7, TB0OUTH, SVMOUT – SD	P4.2, UCB1SOMI, UCB1SCL - SD
GND	P7.7, TB0CLK, MCLK

4. Выполнение работы

3.5 Временные диаграммы SPI

На рисунке желтым показан сигнал CLK, а синим – SIMO.

Считывание данных происходит про фронту CLK. Старшие биты передаются впереди.

На рисунке 4.1 показана команда установки нормального порядка строк: 0xC0.

На рисунке 4.2 показана команда установки зеркального порядка строк: 0xC8.

На рисунке 4.3 показана команда установки адреса страницы 0: 0xB0.

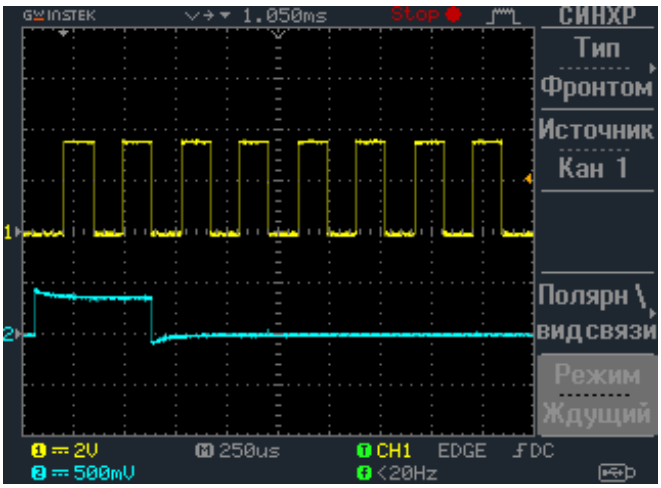

Рисунок 4.1 – Временная диаграммы команды установки нормального порядка строк

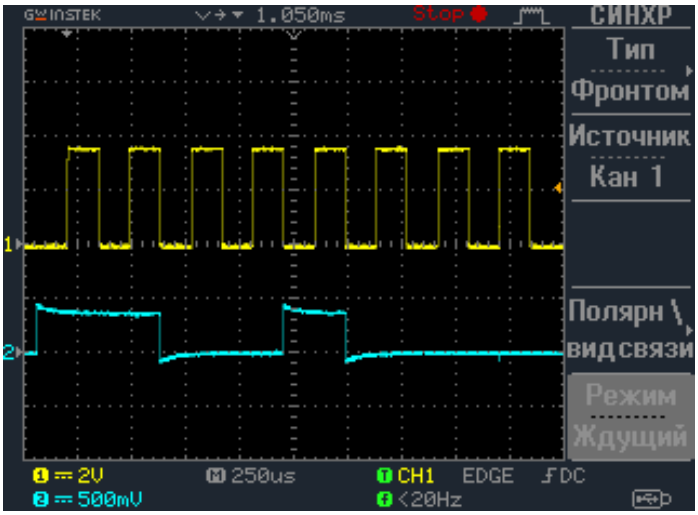

Рисунок 4.2 – Временная диаграммы команды установки зеркального порядка строк

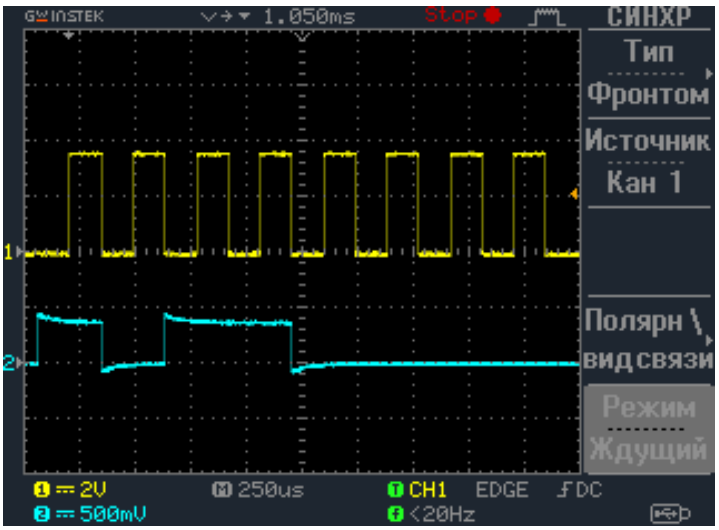

Рисунок 4.3 – Временная диаграммы команды установки страницы

На рисунке 4.4 показана две команды установки адреса столбца 72: 0x08, 0x14.

На рисунке 4.5 показан процесс начала записи данных, который включает установку адреса страницы, столбца и записи 6 байт данных, что соответствует одну символу.

На рисунках 4.6 и 4.7 показан процесс записи 6 байт символа.

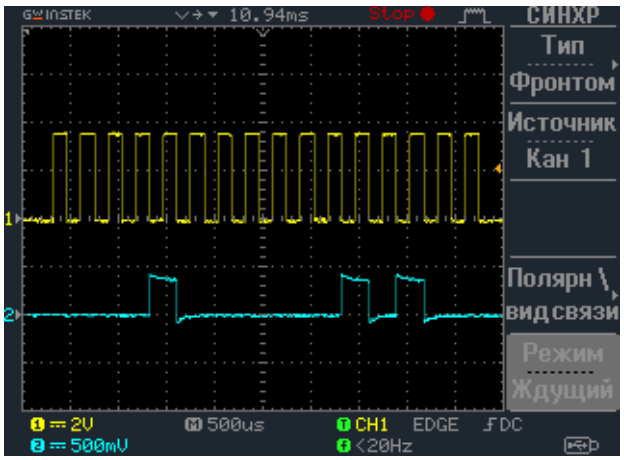

Рисунок 4.4 – Временная диаграммы команд установки адреса столбца

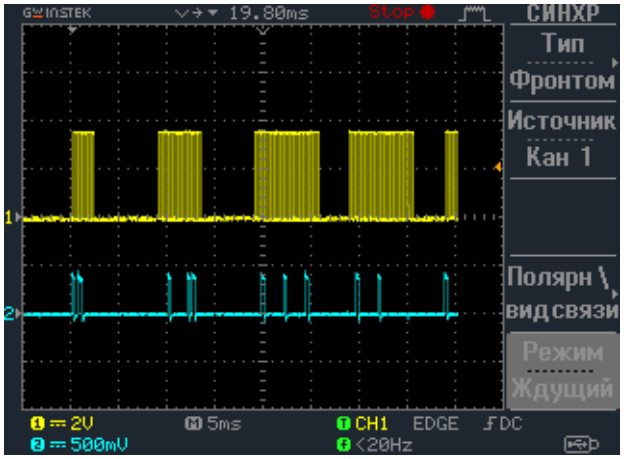

Рисунок 4.5 – Временная диаграммы начала процесса записи данных

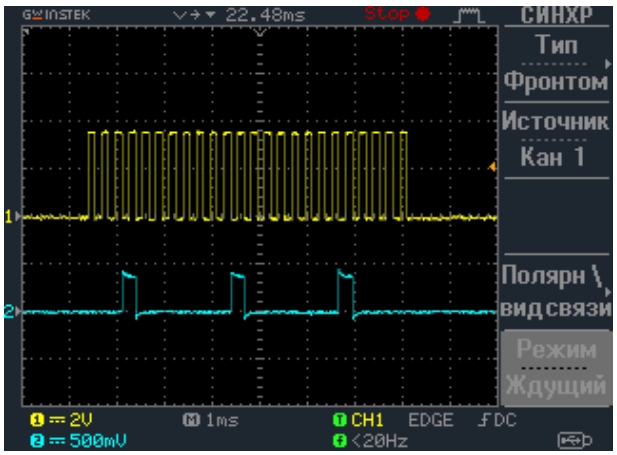

Рисунок 4.6 – Временная диаграммы записи первых трех байт символа

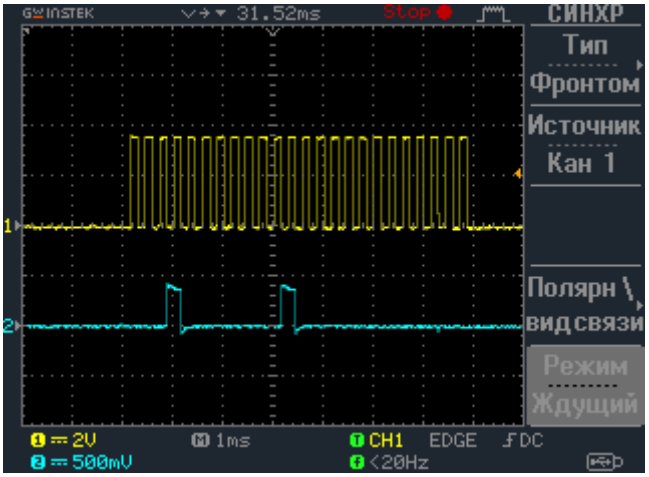

Рисунок 4.7 – Временная диаграммы записи последних трех байт символа

3.6 Листинг кода

```
#include <msp430.h>
#include <stdint.h>
//#include <math.h>

// CONSTANTS
#define MCLK                    25000000
#define TICKSPERUS              (MCLK / 1000000)

// PORT DEFINITIONS
#define ACCEL_INT_IN            P2IN
#define ACCEL_INT_OUT           P2OUT
#define ACCEL_INT_DIR           P2DIR
#define ACCEL_SCK_SEL           P2SEL
#define ACCEL_INT_IE            P2IE
#define ACCEL_INT_IES           P2IES
#define ACCEL_INT_IFG           P2IFG
#define ACCEL_INT_VECTOR        PORT2_VECTOR
#define ACCEL_OUT               P3OUT
#define ACCEL_DIR               P3DIR
#define ACCEL_SEL               P3SEL

// PIN DEFINITIONS
#define ACCEL_INT               BIT5
#define ACCEL_CS                BIT5
#define ACCEL_SIMO              BIT3
#define ACCEL_SOMI              BIT4
#define ACCEL_SCK               BIT7
#define ACCEL_PWR               BIT6

// ACCELEROMETER REGISTER DEFINITIONS
#define REVID                   0x01
#define CTRL                    0x02
#define MODE_400                0x04        // Measurement mode 400 Hz ODR
#define MODE_400_M              0x0C
#define DOUTX                   0x06
#define DOUTY                   0x07
#define DOUTZ                   0x08
#define G_RANGE_2               0x80        // 2g range
#define I2C_DIS                 0x10        // I2C disabled

int8_t accelData;
int8_t RevID;
int8_t Cma3000_xAccel;
```

```
int8_t Cma3000_yAccel;
int8_t Cma3000_zAccel;

// Screen size
#define DOGS102x6_X_SIZE   102         // Display Size in dots: X-Axis
#define DOGS102x6_Y_SIZE    64         // Display Size in dots: Y-Axis

// Screen printing styles
#define DOGS102x6_DRAW_NORMAL   0x00   // Display dark pixels on a light
background
#define DOGS102x6_DRAW_INVERT   0x01   // Display light pixels on a dark
background

// Screen printing mode
#define DOGS102x6_DRAW_IMMEDIATE  0x01  // Display update done immediately
#define DOGS102x6_DRAW_ON_REFRESH 0x00  // Display update done only with refresh

#define SET_COLUMN_ADDRESS_MSB        0x10  //Set SRAM col. addr. before write,
last 4 bits =
                                            // ca4-ca7
#define SET_COLUMN_ADDRESS_LSB        0x00  //Set SRAM col. addr. before write,
last 4 bits =
                                            // ca0-ca3
#define SET_POWER_CONTROL             0x2F  //Set Power control - booster,
regulator, and follower
                                            // on
#define SET_SCROLL_LINE               0x40  //Scroll image up by SL rows (SL =
last 5 bits),
                                            // range:0-63
#define SET_PAGE_ADDRESS              0xB0  //Set SRAM page addr (pa = last 4
bits), range:0-8
#define SET_VLCD_RESISTOR_RATIO       0x27  //Set internal resistor ratio Rb/Ra
to adjust contrast
#define SET_ELECTRONIC_VOLUME_MSB     0x81  //Set Electronic Volume "PM" to
adjust contrast
#define SET_ELECTRONIC_VOLUME_LSB     0x0F  //Set Electronic Volume "PM" to
adjust contrast (PM =
                                            // last 5 bits)
#define SET_ALL_PIXEL_ON              0xA4  //Disable all pixel on (last bit 1 to
turn on all pixels
                                            // - does not affect memory)
#define SET_INVERSE_DISPLAY           0xA6  //Inverse display off (last bit 1 to
invert display -
                                            // does not affect memory)
#define SET_DISPLAY_ENABLE            0xAF  //Enable display (exit sleep mode &
restore power)
#define SET_SEG_DIRECTION             0xA1  //Mirror SEG (column) mapping (set
bit0 to mirror
                                            // display)
#define SET_COM_DIRECTION             0xC8  //Mirror COM (row) mapping (set bit3
to mirror display)

#define SET_SEG_DIRECTION_1           0xA5  //Mirror SEG (column) mapping (set
bit0 to mirror
                                            // display)
#define SET_COM_DIRECTION_1             0xC4  //Mirror COM (row) mapping (set
bit3 to mirror display)

#define SYSTEM_RESET                  0xE2  //Reset the system. Control regs
reset, memory not
                                            // affected
#define NOP                           0xE3  //No operation
#define SET_LCD_BIAS_RATIO            0xA2  //Set voltage bias ratio (BR = bit0)
#define SET_CURSOR_UPDATE_MODE        0xE0  //Column address will increment with
write operation
                                            // (but no wrap around)
```

```
#define RESET_CURSOR_UPDATE_MODE      0xEE  //Return cursor to column address
from before cursor
                                            // update mode was set
#define SET_ADV_PROGRAM_CONTROL0_MSB  0xFA  //Set temp. compensation curve to -
0.11%/C
#define SET_ADV_PROGRAM_CONTROL0_LSB  0x90

// Pins from MSP430 connected to LCD
#define CD              BIT6
#define CS              BIT4
#define RST             BIT7
#define BACKLT          BIT6
#define SPI_SIMO        BIT1
#define SPI_CLK         BIT3

// Ports
#define CD_RST_DIR      P5DIR
#define CD_RST_OUT      P5OUT
#define CS_BACKLT_DIR   P7DIR
#define CS_BACKLT_OUT   P7OUT
#define CS_BACKLT_SEL   P7SEL
#define SPI_SEL         P4SEL
#define SPI_DIR         P4DIR


uint8_t dogs102x6Memory[816 + 2];

uint8_t currentPage = 0, currentColumn = 0;
uint8_t colSize = 6;
int CURRENT_ORIENTATION = 0;

uint8_t backlight  = 8;
uint8_t contrast = 0x0F;
uint8_t drawmode = DOGS102x6_DRAW_IMMEDIATE;

uint8_t Dogs102x6_initMacro[] = {
    SET_SCROLL_LINE,
    SET_SEG_DIRECTION,
    SET_COM_DIRECTION,
    SET_ALL_PIXEL_ON,
    SET_INVERSE_DISPLAY,
    SET_LCD_BIAS_RATIO,
    SET_POWER_CONTROL,
    SET_VLCD_RESISTOR_RATIO,
    SET_ELECTRONIC_VOLUME_MSB,
    SET_ELECTRONIC_VOLUME_LSB,
    SET_ADV_PROGRAM_CONTROL0_MSB,
    SET_ADV_PROGRAM_CONTROL0_LSB,
    SET_DISPLAY_ENABLE,
    SET_PAGE_ADDRESS,
    SET_COLUMN_ADDRESS_MSB,
    SET_COLUMN_ADDRESS_LSB
};

uint8_t Dogs102x6_inv1Macro[] = {
    SET_SEG_DIRECTION,
    SET_COM_DIRECTION
};

uint8_t Dogs102x6_inv2Macro[] = {
    SET_SEG_DIRECTION_1,
    SET_COM_DIRECTION_1
};

void Dogs102x6_writeCommand(uint8_t *sCmd, uint8_t i);
int parseProjectionByte(uint8_t projectionByte);
```

```
int8_t Cma3000_readRegister(int8_t Address);
int8_t Cma3000_writeRegister(uint8_t Address, int8_t accelData);
void ShowNumber(int num);
void SetupButtons();
double atan(double x, int n);


int MAPPING_VALUES[] = { 1142, 571, 286, 143, 71, 36, 18 };
uint8_t BITx[] = { BIT6, BIT5, BIT4, BIT3, BIT2, BIT1, BIT0 };
int num_z=1, num_y=1;

int pos_on_screen = 0;

static const uint8_t FONT6x8[] = {
    /* 6x8 font, each line is a character each byte is a one pixel wide column
     * of that character. MSB is the top pixel of the column, LSB is the bottom
     * pixel of the column. 0 = pixel off. 1 = pixel on. */

    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // space
    0x00, 0x00, 0xFA, 0x00, 0x00, 0x00, // !
    0x00, 0xE0, 0x00, 0xE0, 0x00, 0x00, // "
    0x28, 0xFE, 0x28, 0xFE, 0x28, 0x00, // #
    0x24, 0x54, 0xFE, 0x54, 0x48, 0x00, // $
    0xC4, 0xC8, 0x10, 0x26, 0x46, 0x00, // %
    0x6C, 0x92, 0x6A, 0x04, 0x0A, 0x00, // &
    0x00, 0x10, 0xE0, 0xC0, 0x00, 0x00, // '
    0x00, 0x38, 0x44, 0x82, 0x00, 0x00, // (
    0x00, 0x82, 0x44, 0x38, 0x00, 0x00, // )
    0x54, 0x38, 0xFE, 0x38, 0x54, 0x00, // *
    0x10, 0x10, 0x7C, 0x10, 0x10, 0x00, // +
    0x00, 0x02, 0x1C, 0x18, 0x00, 0x00, // ,
    0x10, 0x10, 0x10, 0x10, 0x10, 0x00, // -
    0x00, 0x00, 0x06, 0x06, 0x00, 0x00, // .
    0x04, 0x08, 0x10, 0x20, 0x40, 0x00, // /
    //96 Bytes
    0x7C, 0x8A, 0x92, 0xA2, 0x7C, 0x00, // 0
    0x00, 0x42, 0xFE, 0x02, 0x00, 0x00, // 1
    0x42, 0x86, 0x8A, 0x92, 0x62, 0x00, // 2
    0x84, 0x82, 0x92, 0xB2, 0xCC, 0x00, // 3
    0x18, 0x28, 0x48, 0xFE, 0x08, 0x00, // 4
    0xE4, 0xA2, 0xA2, 0xA2, 0x9C, 0x00, // 5
    0x3C, 0x52, 0x92, 0x92, 0x0C, 0x00, // 6
    0x82, 0x84, 0x88, 0x90, 0xE0, 0x00, // 7
    0x6C, 0x92, 0x92, 0x92, 0x6C, 0x00, // 8
    0x60, 0x92, 0x92, 0x94, 0x78, 0x00, // 9

    0x00, 0x7C, 0xA2, 0x92, 0x8A, 0x7C, // 0 inv
    //0x7C, 0x8A, 0x92, 0xA2, 0x7C, 0x00, // 0
    0x00, 0x02, 0x02, 0xFE, 0x42, 0x00, // 1 inv
    //0x00, 0x42, 0xFE, 0x02, 0x00, 0x00, // 1
    0x00, 0x62, 0x92, 0x8A, 0x86, 0x42, // 2 inv
    //0x42, 0x86, 0x8A, 0x92, 0x62, 0x00, // 2
    0x00, 0xCC, 0xB2, 0xB92, 0x82, 0x84, // 3 inv
    //0x84, 0x82, 0x92, 0xB2, 0xCC, 0x00, // 3
    0x00, 0x08, 0xFE, 0x48, 0x28, 0x18, // 4 inv
    //0x18, 0x28, 0x48, 0xFE, 0x08, 0x00, // 4
    0x00, 0x9C, 0xA2, 0xA2, 0xA2, 0xE4, // 5 inv
    //0xE4, 0xA2, 0xA2, 0xA2, 0x9C, 0x00, // 5 inv
    0x00, 0x0C, 0x92, 0x92, 0x52, 0x3C, // 6 inv
    //0x3C, 0x52, 0x92, 0x92, 0x0C, 0x00, // 6 inv
    0x00, 0xE0, 0x90, 0x88, 0x84, 0x82, // 7 inv
    //0x82, 0x84, 0x88, 0x90, 0xE0, 0x00, // 7 inv
    0x00, 0x6C, 0x92, 0x92, 0x92, 0x6C, // 8 inv
    //0x6C, 0x92, 0x92, 0x92, 0x6C, 0x00, // 8 inv
    0x00, 0x78, 0x94, 0x92, 0x92, 0x60, // 9 inv
    //0x60, 0x92, 0x92, 0x94, 0x78, 0x00, // 9 inv
```

```
    0x00, 0x10, 0x10, 0x7C , 0x10, 0x10, // + inv
    // 0x10, 0x10, 0x7C, 0x10, 0x10, 0x00, // +
    0x00, 0x10, 0x10, 0x10, 0x10, 0x10, // - inv
    //0x10, 0x10, 0x10, 0x10, 0x10, 0x00, // -
};


void Dogs102x6_init(void)
{
    // Port initialization for LCD operation
    CD_RST_DIR |= RST;
    // Reset is active low
    CD_RST_OUT &= RST;
    // Reset is active low
    CD_RST_OUT |= RST;
    // Chip select for LCD
    CS_BACKLT_DIR |= CS;
    // CS is active low
    CS_BACKLT_OUT &= ~CS;
    // Command/Data for LCD
    CD_RST_DIR |= CD;
    // CD Low for command
    CD_RST_OUT &= ~CD;

    // P4.1 option select SIMO
    SPI_SEL |= SPI_SIMO;
    SPI_DIR |= SPI_SIMO;
    // P4.3 option select CLK
    SPI_SEL |= SPI_CLK;
    SPI_DIR |= SPI_CLK;

    // Initialize USCI_B1 for SPI Master operation
    // Put state machine in reset
    UCB1CTL1 |= UCSWRST;
    //3-pin, 8-bit SPI master
    UCB1CTL0 = UCCKPH + UCMSB + UCMST + UCMODE_0 + UCSYNC;
    // Clock phase - data captured first edge, change second edge
    // MSB
    // Use SMCLK, keep RESET
    UCB1CTL1 = UCSSEL_2 + UCSWRST;
    UCB1BR0 = 0x02;
    UCB1BR1 = 0;
    // Release USCI state machine
    UCB1CTL1 &= ~UCSWRST;
    UCB1IFG &= ~UCRXIFG;

    Dogs102x6_writeCommand(Dogs102x6_initMacro, 13);

    // Deselect chip
    CS_BACKLT_OUT |= CS;

    dogs102x6Memory[0] = 102;
    dogs102x6Memory[1] = 8;
}

void Dogs102x6_writeCommand(uint8_t *sCmd, uint8_t i)
{
    // Store current GIE state
    uint16_t gie = __get_SR_register() & GIE;

    // Make this operation atomic
    __disable_interrupt();

    // CS Low
    P7OUT &= ~CS;
```

```
    // CD Low
    P5OUT &= ~CD;
    while (i)
    {
        // USCI_B1 TX buffer ready?
        while (!(UCB1IFG & UCTXIFG)) ;

        // Transmit data
        UCB1TXBUF = *sCmd;

        // Increment the pointer on the array
        sCmd++;

        // Decrement the Byte counter
        i--;
    }

    // Wait for all TX/RX to finish
    while (UCB1STAT & UCBUSY) ;

    // Dummy read to empty RX buffer and clear any overrun conditions
    UCB1RXBUF;

    // CS High
    P7OUT |= CS;

    // Restore original GIE state
    __bis_SR_register(gie);
}

void Dogs102x6_writeData(uint8_t *sData, uint8_t i)
{
    // Store current GIE state
    uint16_t gie = __get_SR_register() & GIE;

    // Make this operation atomic
    __disable_interrupt();

    if (drawmode == DOGS102x6_DRAW_ON_REFRESH)
    {
      while (i)
      {
          dogs102x6Memory[2 + (currentPage * 102) + currentColumn] =
(uint8_t)*sData++;
          currentColumn++;

          // Boundary check
          if (currentColumn > 101)
          {
              currentColumn = 101;
          }

          // Decrement the Byte counter
          i--;
      }
    }
    else
    {

      // CS Low
      P7OUT &= ~CS;
      //CD High
      P5OUT |= CD;

      while (i)
      {
```

```
          dogs102x6Memory[2 + (currentPage * 102) + currentColumn] =
(uint8_t)*sData;
          currentColumn++;

          // Boundary check
          if (currentColumn > 101)
          {
              currentColumn = 101;
          }

          // USCI_B1 TX buffer ready?
          while (!(UCB1IFG & UCTXIFG)) ;

          // Transmit data and increment pointer
          UCB1TXBUF = *sData++;

          // Decrement the Byte counter
          i--;
      }

      // Wait for all TX/RX to finish
      while (UCB1STAT & UCBUSY) ;

      // Dummy read to empty RX buffer and clear any overrun conditions
      UCB1RXBUF;

      // CS High
      P7OUT |= CS;
    }

    // Restore original GIE state
    __bis_SR_register(gie);
}

void Dogs102x6_setAddress(uint8_t pa, uint8_t ca)
{
    uint8_t cmd[1];

    // Page boundary check
    if (pa > 7)
    {
        pa = 7;
    }

    // Column boundary check
    if (ca > 101)
    {
        ca = 101;
    }

    // Page Address Command = Page Address Initial Command + Page Address
    cmd[0] = SET_PAGE_ADDRESS + (7 - pa);
    uint8_t H = 0x00;
    uint8_t L = 0x00;
    uint8_t ColumnAddress[] = { SET_COLUMN_ADDRESS_MSB, SET_COLUMN_ADDRESS_LSB };

    currentPage = pa;
    currentColumn = ca;

    if (drawmode == DOGS102x6_DRAW_ON_REFRESH) return; // exit if drawmode on
refresh

    // Separate Command Address to low and high
    L = (ca & 0x0F);
    H = (ca & 0xF0);
    H = (H >> 4);
```

```
    // Column Address CommandLSB = Column Address Initial Command
    //                           + Column Address bits 0..3
    ColumnAddress[0] = SET_COLUMN_ADDRESS_LSB + L;
    // Column Address CommandMSB = Column Address Initial Command
    //                           + Column Address bits 4..7
    ColumnAddress[1] = SET_COLUMN_ADDRESS_MSB + H;

    // Set page address
    Dogs102x6_writeCommand(cmd, 1);
    // Set column address
    Dogs102x6_writeCommand(ColumnAddress, 2);
}

void Dogs102x6_clearScreen(void)
{
    uint8_t LcdData[] = {0x00};
    uint8_t p, c;

    // 8 total pages in LCD controller memory
    for (p = 0; p < 8; p++)
    {
        Dogs102x6_setAddress(p, 0);
        // 102 total columns in LCD controller memory
        for (c = 0; c < 102; c++)
        {
            Dogs102x6_writeData(LcdData, 1);
        }
    }
}

void Dogs102x6_charDraw(uint8_t row, uint8_t col, uint16_t f, uint8_t style)
{
    // Each Character consists of 6 Columns on 1 Page
    // Each Page presents 8 pixels vertically (top = MSB)
    uint8_t b;
    uint16_t h;
    uint8_t inverted_char[6];

    // Row boundary check
    if (row > 7)
    {
        row = 7;
    }

    // Column boundary check
    if (col > 101)
    {
        col = 101;
    }

    // handle characters not in our table
    if (f < 32 || f > 129)
    {
        // replace the invalid character with a '.'
        f = '.';
    }

    // subtract 32 because FONT6x8[0] is "space" which is ascii 32,
    // multiply by 6 because each character is columns wide
    h = (f - 32) * 6;

    Dogs102x6_setAddress(row, col);
    if (style == DOGS102x6_DRAW_NORMAL)
    {
        // write character
        Dogs102x6_writeData((uint8_t *)FONT6x8 + h, 6);
```

```
    }
    else
    {
        for (b = 0; b < 6; b++)
        {
            // invert the character
            inverted_char[b] = FONT6x8[h + b] ^ 0xFF;
        }
        // write inverted character
        Dogs102x6_writeData(inverted_char, 6);
    }
}

void Dogs102x6_backlightInit(void)
{
    // Turn on Backlight
    CS_BACKLT_DIR |= BACKLT;
    CS_BACKLT_OUT |= BACKLT;
    // Uses PWM to control brightness
    CS_BACKLT_SEL |= BACKLT;

    // start at full brightness (8)
    TB0CCTL4 = OUTMOD_7;
    TB0CCR4 = TB0CCR0 >> 1;

    TB0CCR0 = 50;
    TB0CTL = TBSSEL_1 + MC_1;
}


void Dogs102x6_setInverseDisplay(void)
{
    uint8_t cmd[] = {SET_INVERSE_DISPLAY + 0x01};

    Dogs102x6_writeCommand(cmd, 1);
}

void Dogs102x6_setBacklight(uint8_t brightness)
{
    unsigned int dutyCycle = 0, i, dummy;

    if (brightness > 0)
    {
        TB0CCTL4 = OUTMOD_7;
        dummy = (TB0CCR0 >> 4);

        dutyCycle = 12;
        for (i = 0; i < brightness; i++)
            dutyCycle += dummy;

        TB0CCR4 = dutyCycle;

        //If the backlight was previously turned off, turn it on.
        if (!backlight)
            TB0CTL |= MC0;
    }
    else
    {
        TB0CCTL4 = 0;
        TB0CTL &= ~MC0;
    }
    backlight = brightness;
}

void Cma3000_init(void)
{
```

```
    do
    {
        // Set P3.6 to output direction high
        ACCEL_OUT |= ACCEL_PWR;
        ACCEL_DIR |= ACCEL_PWR;

        // P3.3,4 option select
        ACCEL_SEL |= ACCEL_SIMO + ACCEL_SOMI;

        // P2.7 option select
        ACCEL_SCK_SEL |= ACCEL_SCK;

        ACCEL_INT_DIR &= ~ACCEL_INT;

        // Generate interrupt on Lo to Hi edge
        ACCEL_INT_IES &= ~ACCEL_INT;

        // Clear interrupt flag
        ACCEL_INT_IFG &= ~ACCEL_INT;

        // Unselect acceleration sensor
        ACCEL_OUT |= ACCEL_CS;
        ACCEL_DIR |= ACCEL_CS;

        // **Put state machine in reset**
        UCA0CTL1 |= UCSWRST;
        // 3-pin, 8-bit SPI master Clock polarity high, MSB
        UCA0CTL0 = UCMST + UCSYNC + UCCKPH + UCMSB;
        // Use SMCLK, keep RESET
        UCA0CTL1 = UCSWRST + UCSSEL_2;
        // /0x30
        UCA0BR0 = 0x30;
        // 0
        UCA0BR1 = 0;
        // No modulation
        UCA0MCTL = 0;
        // **Initialize USCI state machine**
        UCA0CTL1 &= ~UCSWRST;

        // Read REVID register
        RevID = Cma3000_readRegister(REVID);
        __delay_cycles(50 * TICKSPERUS);

        // Activate measurement mode: 2g/400Hz
        accelData = Cma3000_writeRegister(CTRL, G_RANGE_2 | I2C_DIS | MODE_400
/*MODE_400_M*/);

        // Settling time per DS = 10ms
        __delay_cycles(1000 * TICKSPERUS);

        // INT pin interrupt disabled
        ACCEL_INT_IE  &= ~ACCEL_INT;

        // Repeat till interrupt Flag is set to show sensor is working
    } while (!(ACCEL_INT_IN & ACCEL_INT));
}

int8_t Cma3000_writeRegister(uint8_t Address, int8_t accelData)
{
    uint8_t Result;

    // Address to be shifted left by 2
    Address <<= 2;

    // RW bit to be set
    Address |= 2;
```

```
    // Select acceleration sensor
    ACCEL_OUT &= ~ACCEL_CS;

    // Read RX buffer just to clear interrupt flag
    Result = UCA0RXBUF;

    // Wait until ready to write
    while (!(UCA0IFG & UCTXIFG)) ;

    // Write address to TX buffer
    UCA0TXBUF = Address;

    // Wait until new data was written into RX buffer
    while (!(UCA0IFG & UCRXIFG)) ;

    // Read RX buffer just to clear interrupt flag
    Result = UCA0RXBUF;

    // Wait until ready to write
    while (!(UCA0IFG & UCTXIFG)) ;

    // Write data to TX buffer
    UCA0TXBUF = accelData;

    // Wait until new data was written into RX buffer
    while (!(UCA0IFG & UCRXIFG)) ;

    // Read RX buffer
    Result = UCA0RXBUF;

    // Wait until USCI_A0 state machine is no longer busy
    while (UCA0STAT & UCBUSY) ;

    // Deselect acceleration sensor
    ACCEL_OUT |= ACCEL_CS;

    return Result;
}

uint8_t CMA3000_writeCommand(uint8_t firstByte, uint8_t secondByte) {
    char indata;

    P3OUT &= ~BIT5;

    indata = UCA0RXBUF;

    while(!(UCA0IFG & UCTXIFG));

    UCA0TXBUF = firstByte;

    while(!(UCA0IFG & UCRXIFG));

    indata = UCA0RXBUF;

    while(!(UCA0IFG & UCTXIFG));

    UCA0TXBUF = secondByte;

    while(!(UCA0IFG & UCRXIFG));

    indata = UCA0RXBUF;

    while(UCA0STAT & UCBUSY);

    P3OUT |= BIT5;
```

```
    return indata;
}


int8_t Cma3000_readRegister(int8_t Address)
{
    int8_t Result;

    // Address to be shifted left by 2 and RW bit to be reset
    Address <<= 2;

    // Select acceleration sensor
    ACCEL_OUT &= ~ACCEL_CS;

    // Read RX buffer just to clear interrupt flag
    Result = UCA0RXBUF;

    // Wait until ready to write
    while (!(UCA0IFG & UCTXIFG)) ;

    // Write address to TX buffer
    UCA0TXBUF = Address;

    // Wait until new data was written into RX buffer
    while (!(UCA0IFG & UCRXIFG)) ;

    // Read RX buffer just to clear interrupt flag
    Result = UCA0RXBUF;

    // Wait until ready to write
    while (!(UCA0IFG & UCTXIFG)) ;

    // Write dummy data to TX buffer
    UCA0TXBUF = 0;

    // Wait until new data was written into RX buffer
    while (!(UCA0IFG & UCRXIFG)) ;

    // Read RX buffer
    Result = UCA0RXBUF;

    // Wait until USCI_A0 state machine is no longer busy
    while (UCA0STAT & UCBUSY) ;

    // Deselect acceleration sensor
    ACCEL_OUT |= ACCEL_CS;

    // Return new data from RX buffer
    return Result;
}

int parseProjectionByte(uint8_t projectionByte) {
    int i = 0;
    int projectionValue = 0;

    int isNegative = projectionByte & BIT7;

    for (; i < 7; i++) {
          if (isNegative) {
                projectionValue += (BITx[i] & projectionByte) ? 0 :
MAPPING_VALUES[i];
          }
          else {
                projectionValue += (BITx[i] & projectionByte) ? MAPPING_VALUES[i]
: 0;
```

```
            }
        }

        projectionValue *= isNegative ? -1 : 1;

        return projectionValue;
}

void Cma3000_readAccel(void)
{
        // Read DOUTX register
        Cma3000_xAccel = Cma3000_readRegister(DOUTX);
        __delay_cycles(50 * TICKSPERUS);

        // Read DOUTY register
        Cma3000_yAccel = Cma3000_readRegister(DOUTY);
        __delay_cycles(50 * TICKSPERUS);


        // Read DOUTZ register
        Cma3000_zAccel = Cma3000_readRegister(DOUTZ);
        num_z = parseProjectionByte(Cma3000_zAccel);

        ShowNumber(num_z);
}


#pragma vector=WDT_VECTOR
__interrupt void WDTINT()
{
        SFRIE1 &=~ WDTIE;
        Cma3000_readAccel();
        SFRIE1 |= WDTIE;
}


#pragma vector = PORT2_VECTOR
__interrupt void buttonS1(void)
{
        volatile int i = 0;

        for (i = 0; i < 2000; i++);

        if ((P2IN & BIT2) == 0) {
              if (pos_on_screen == 0) {
                    pos_on_screen = 1;
              }
              else {
                    pos_on_screen = 0;
              }

              Dogs102x6_clearScreen();

              for (i = 0; i < 2000; i++);
        }

        P2IFG = 0;
}

double atan(double x, int n) {
        double a = x;
        double sum = a;
        double b = a;
        double E = 1. / n;
        int i = 1;
        for (i = 1; a > E; i++) {
```

```
        b *= -x * x;
        a *= b / (2 * i + 1);
        sum += a;
    }
    return sum;
}

void ShowNumber(int num)
{

    Dogs102x6_clearScreen();

    volatile int length = 1;
    volatile int digit = 0;
    volatile int j = 0;
    volatile int i = 10;

    while(1)
    {
          if(num / i != 0)
          {
                i *= 10;
                length++;
          }
          else
          {
                break;
          }
    }


    int temp = (int)num;
    temp = fabs_s(num);
    if(pos_on_screen == 0){
          for(j = 0; j < length; j++)
          {
                digit = (int)(temp % 10);

                temp = temp / 10;

                if (digit < 10){
                      Dogs102x6_charDraw(7, length*colSize-j*colSize, 48 + digit
,0);
                }
          }
          if (num < 0)
          {
                Dogs102x6_charDraw(7, 0, 45 ,0);
          }
          else
          {
                Dogs102x6_charDraw(7, 0, 43 ,0);
          }
    }
    else{
          if (num < 0)
          {
                Dogs102x6_charDraw(7, 102-colSize, 48 + 9 + 10 + 2 ,0);
          }
          else
          {
                Dogs102x6_charDraw(7, 102-colSize, 48 + 9 + 10 + 1 ,0);
          }
          for(j = 1; j < length+1; j++)
          {
                digit = (int)(temp % 10);
```

```
                temp = temp / 10;

                if (digit < 10){
                      Dogs102x6_charDraw(7, 102 - (length*colSize-j*colSize)-
colSize-colSize, 48 + 10 + digit ,0);
                }
          }
    }
}

int fabs_s(int num){
    if(num > 0){
          return num;
    }
    else{
          return num*(-1);
    }
}


void SetupButtons()
{

    P4SEL |= BIT2;
    P4DIR |= BIT2;          //Выход с платы
    P4SEL |= BIT1;
    P4DIR |= BIT1;          //Выход с платы
    P4SEL |= BIT0;
    P4DIR |= BIT0;
    P4SEL |= BIT3;
    P4DIR |= BIT3;
    P3DIR |= BIT7;          //Выход с платы
    P3SEL |= BIT7;
    // set buttons for reading
    //P1DIR &= ~BIT7;
    P2DIR &= ~BIT2;
    // Set up pull up resistors
    //P1REN |= BIT7;
    P2REN |= BIT2;
    // initialize buttons with inactive level
    //P1OUT |= BIT7;
    P2OUT |= BIT2;
    // set up interrupts for S1
    //P1IE |= BIT7;
    //P1IES |= BIT7;
    //P1IFG = 0;
    P2IE |= BIT2;
    P2IES |= BIT2;
    P2IFG = 0;
}

/*
 * main.c
 */
int main(void) {
    WDTCTL = WDT_ADLY_250;  // Stop watchdog timer
    SFRIE1 |= WDTIE;
    SetupButtons();

    __bis_SR_register(GIE);
    Cma3000_init();
    Dogs102x6_init();

    Dogs102x6_backlightInit();
    Dogs102x6_setBacklight(5);
```

```
    Dogs102x6_clearScreen();


    __no_operation();


    return 0;
}
```

4 Выводы

В ходе лабораторной работы были изучены принципы организации последовательного интерфейса SPI и подключения устройств на его основе на базе микроконтроллера MSP430F5529. В результате выполнения работы была написана программа, выводящие данные акселерометра на ЖКИ и проанализированы диаграмма работы интерфейса SPI ЖКИ