

Белорусский Государственный Университет
Информатики и Радиоэлектроники
Кафедра ЭВМ

Отчет по лабораторной работе № 4

Тема: «Аналоговый ввод-вывод. АЦП. Компаратор. Потенциометр.
Сенсорная клавиатура»

Выполнили:

ст. гр. 950503
Зарубо Д. Ю
Ященко В.П

Проверил:

Шеменков В.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.

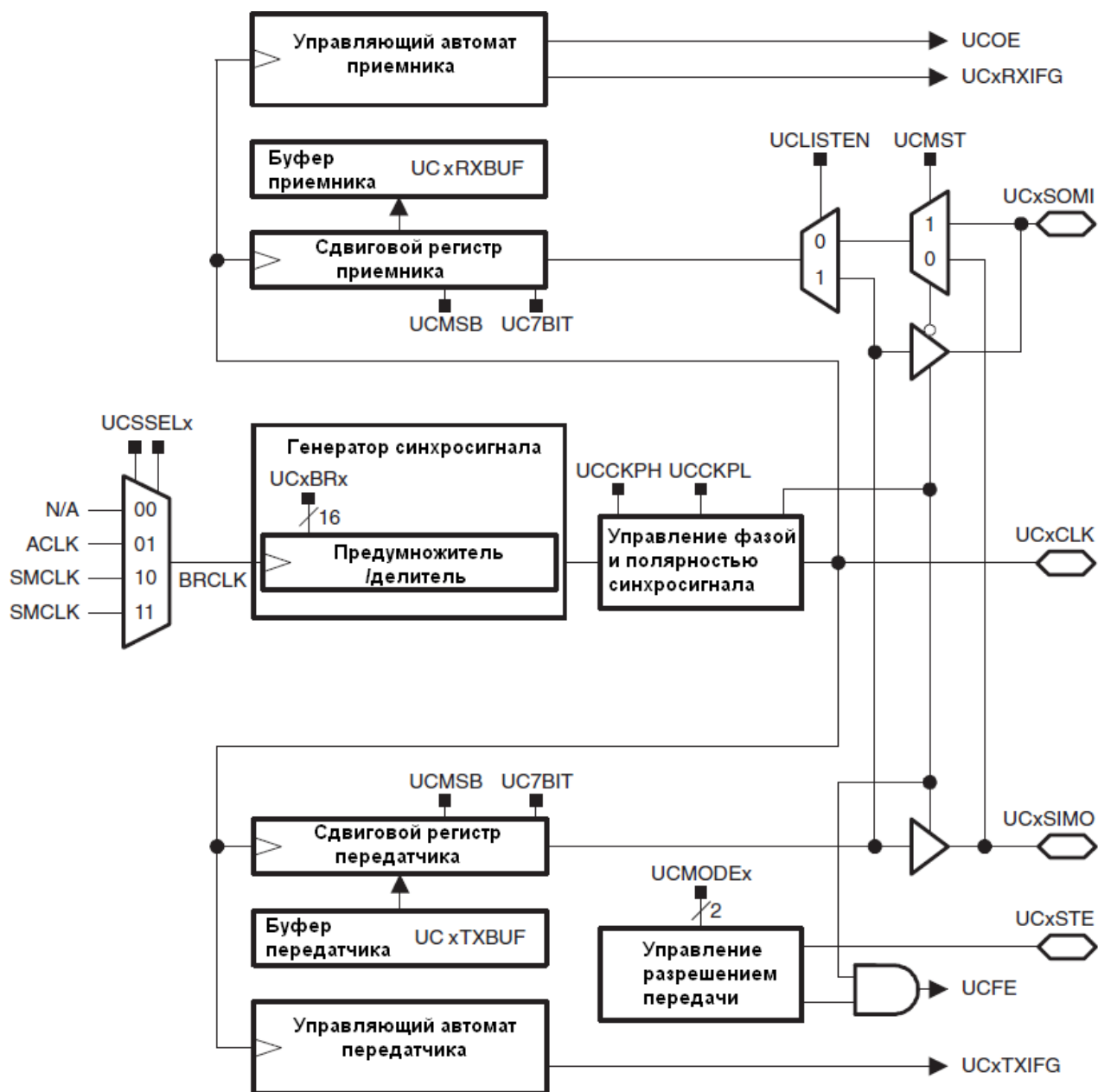


Рисунок 3.1 – Структура интерфейса SPI

Схема передачи данных начинает работу при помещении данных в буферный регистр передатчика UCxTXBUF. Данные автоматически помещаются в сдвиговой регистр (если он пуст), что начинает передачу по линии UCxSIMO. Флаг прерывания UCTXIFG устанавливается при перемещении данных в сдвиговой регистр и сигнализирует об освобождении буферного регистра, а не об окончании 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).

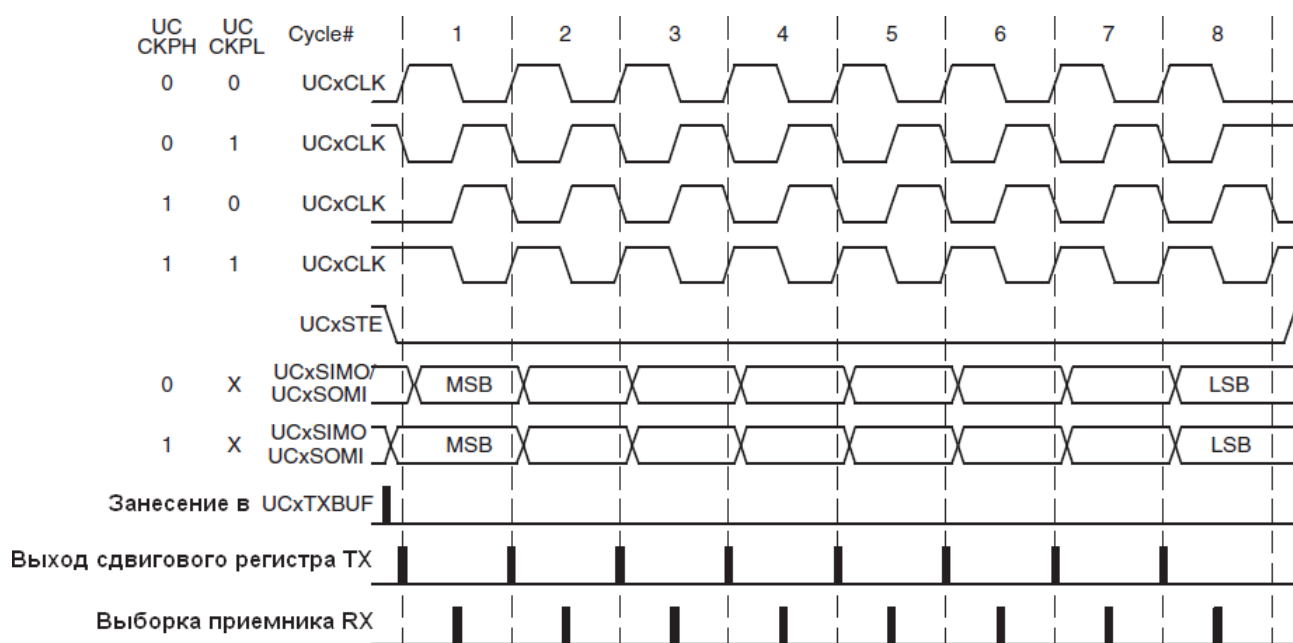


Рисунок 3.2 – Режимы синхронизации

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: $f_{\text{BitClock}} = f_{\text{BRCLK}} / \text{UCBRx}$.

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 (А или В) 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	UCMODE _x	Синхронный режим: 00 – 3pin SPI, 01 – 4pin SPI + STE активный высокий, 10 – 4pin SPI + STE активный низкий, 11 – I ² C	UCMODE_0 ... UCMODE_3
	0	UCSYNC	Режим: синхронный - 1	UCSYNC
UCAxCTL1	6-7	UCSSEL _x	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	UCBR _x	Младший байт делителя частоты	UCA0BR0
UCAxBR1	0-7	UCBR _x	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	UCRXBUF _x	Буфер приемника	UCA0RXBUF
UCAxTXBUF	0-7	UCTXBUF _x	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	UCIV _x	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 последовательный режим SPI, поддерживает чтение данных (в SPI режиме только запись). Устройство содержит двухпортовую статическую DDRAM.

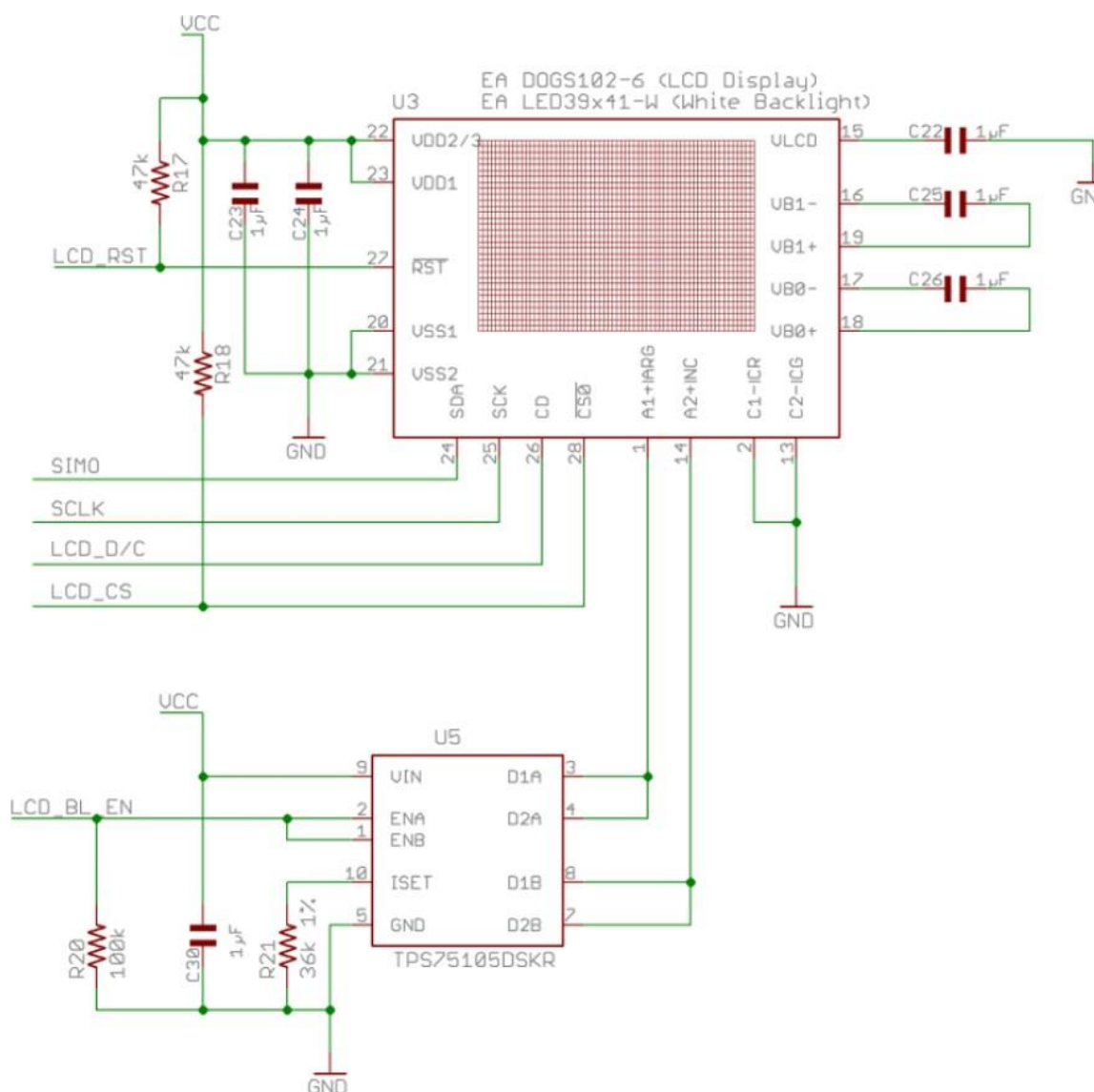


Рисунок 3.3 – Схема подключения ЖКИ экрана

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.

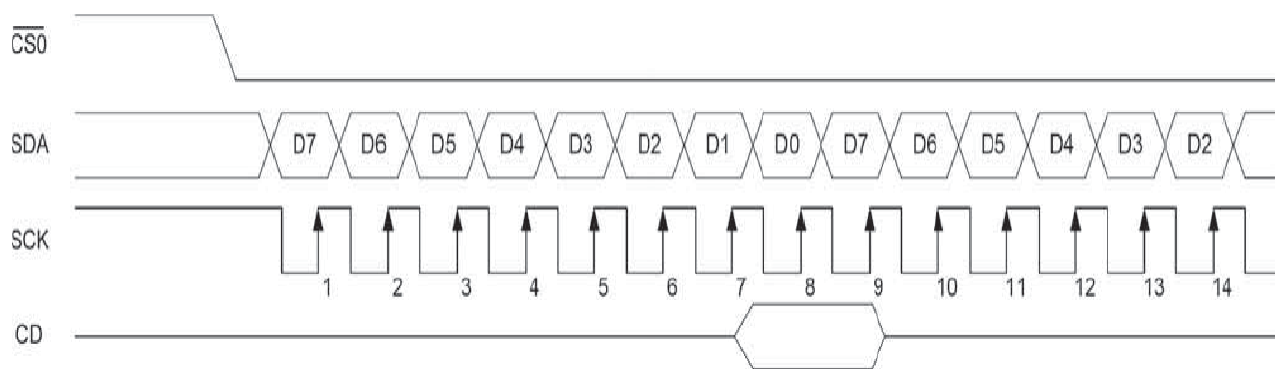


Рисунок 3.4 Временная диаграмма обмена с ЖКИ

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	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.

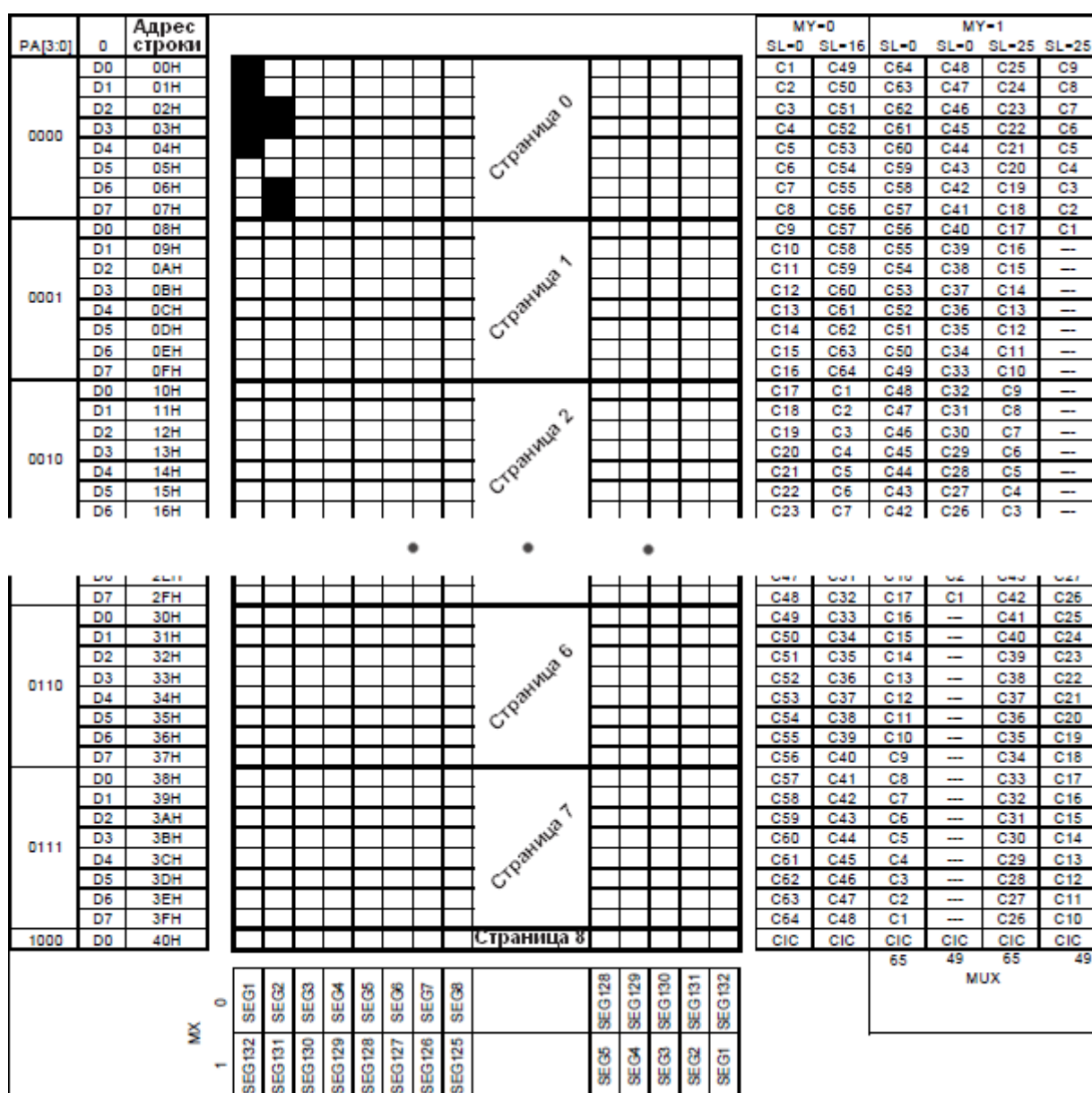


Рисунок 3.5 – Режимы ориентации экрана и вывода строк и столбцов

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.

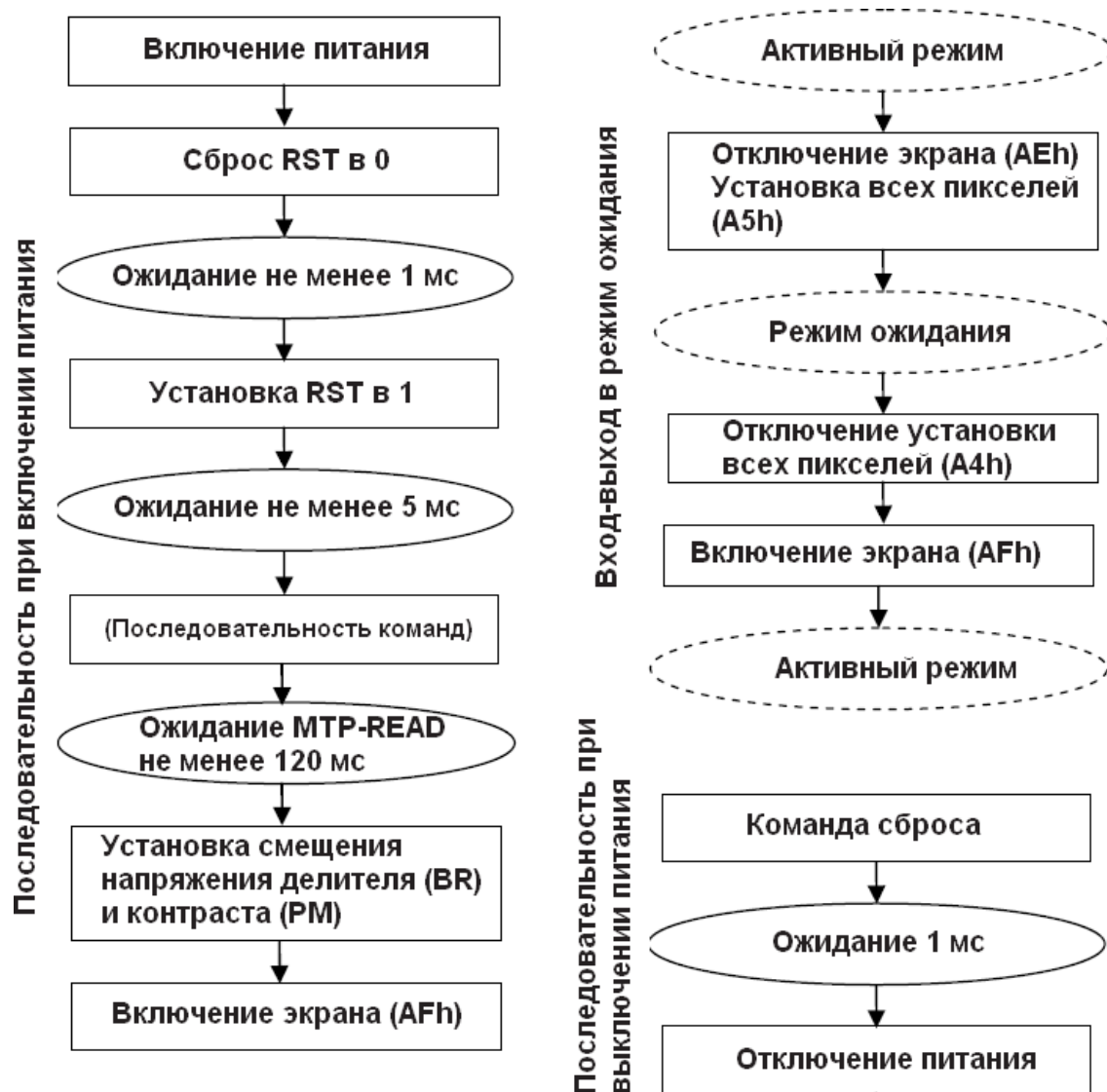


Рисунок 3.7 – Последовательность действий при включении/выключении ЖКИ и при входе/выходе в режим ожидания

3.3 Акселерометр СМА3000-D01

3-координатный акселерометр с цифровым выходом СМА3000-D01 обладает следующими возможностями:

- диапазон измерений задается программно (2g, 8g);
- питание 1.7 — 3.6 В;
- интерфейс SPI или I²C 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.

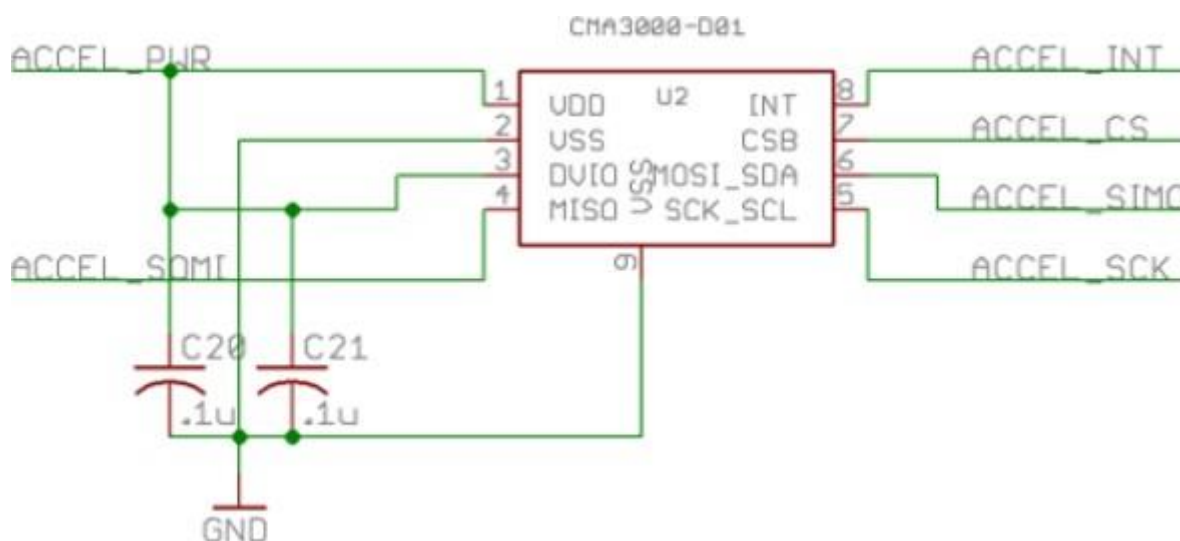


Рисунок 3.8 – Схема подключения акселерометра

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 8г 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 ² C

Выбор интерфейса (SPI или I²C) осуществляется при помощи сигнала выбора кристалла, при этом I²C 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	² Выбор интерфейса I ² C: 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.

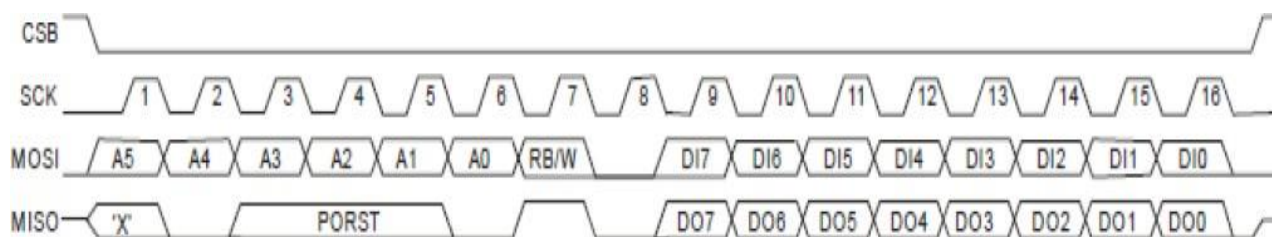


Рисунок 3.10 – Формат фрейма

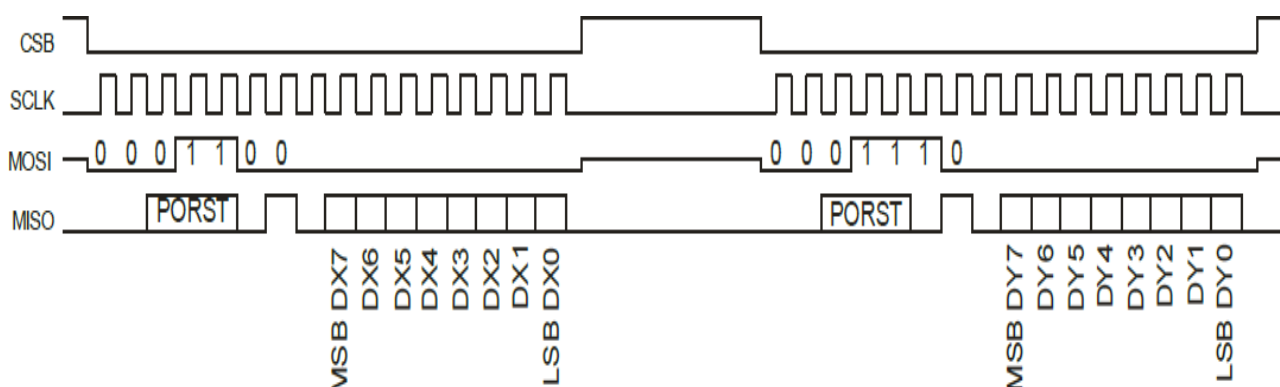


Рисунок 3.11 – Пример операции чтения данных

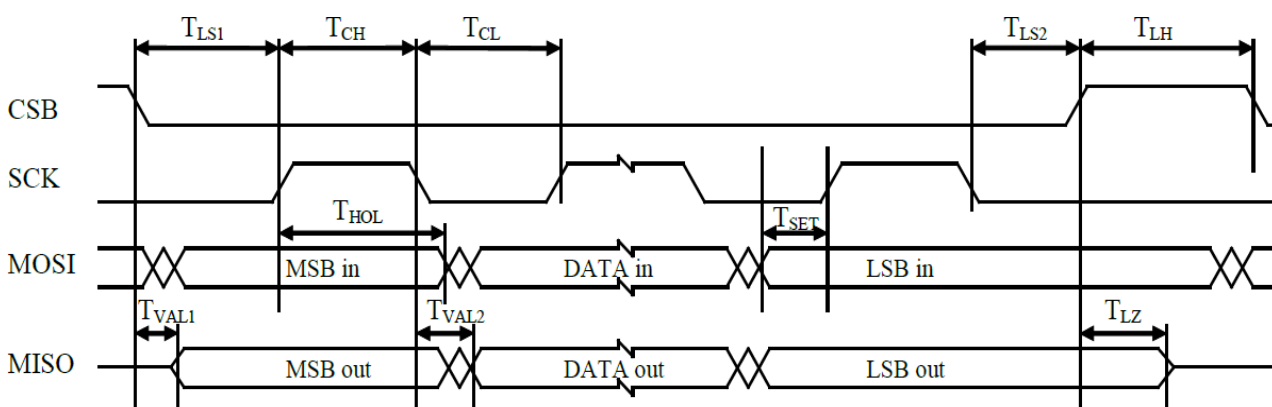


Рисунок 3.12 – Временные параметры обмена

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.

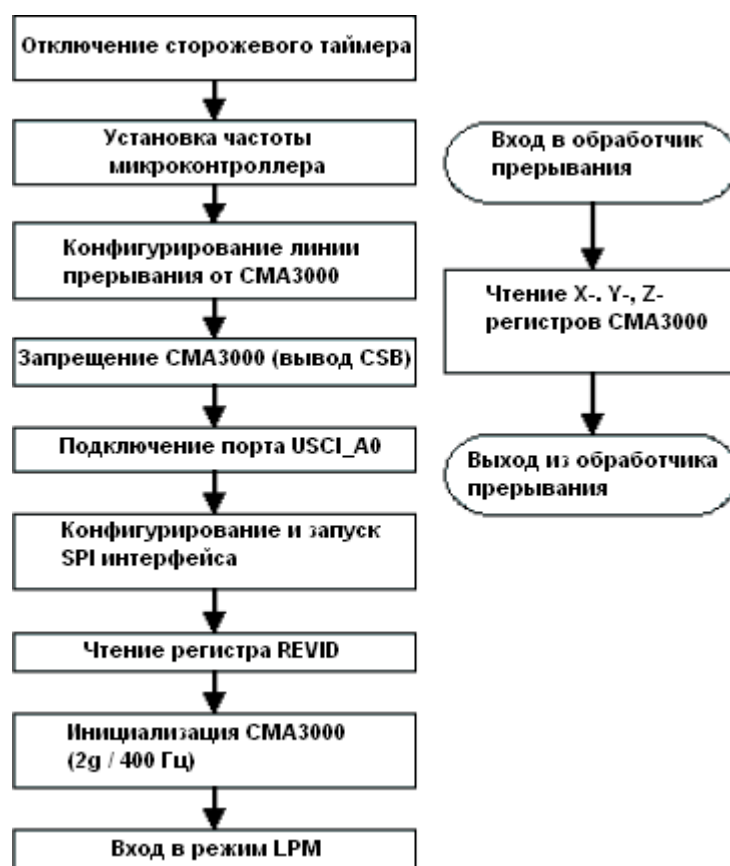


Рисунок 3.13 – Типичная последовательность при инициализации CMA3000-D01

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

1.1 Временные диаграммы SPI

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

Считывание данных происходит по 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.

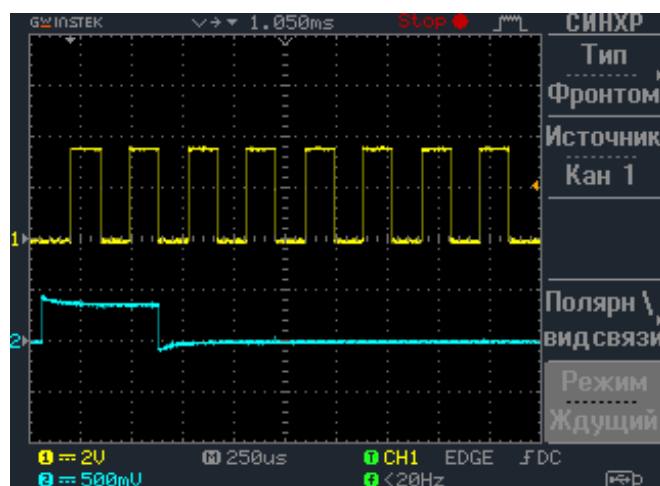


Рисунок 4.1 – Временная диаграммы команды установки нормального порядка строк

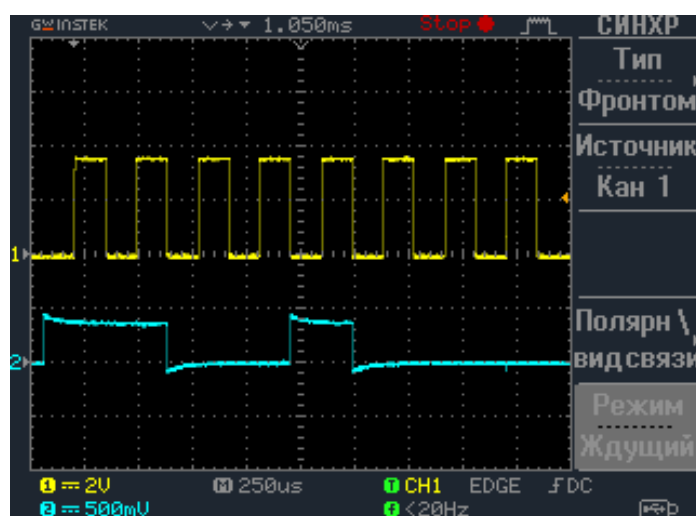


Рисунок 4.2 – Временная диаграммы команды установки зеркального порядка строк

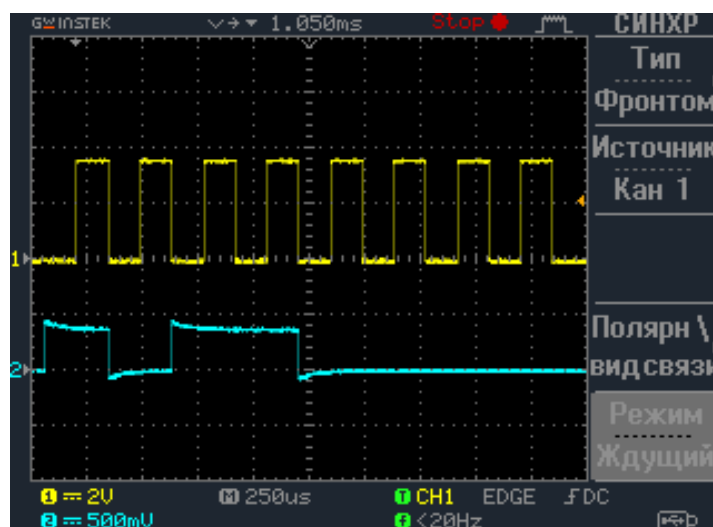


Рисунок 4.3 – Временная диаграммы команды установки страницы

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.

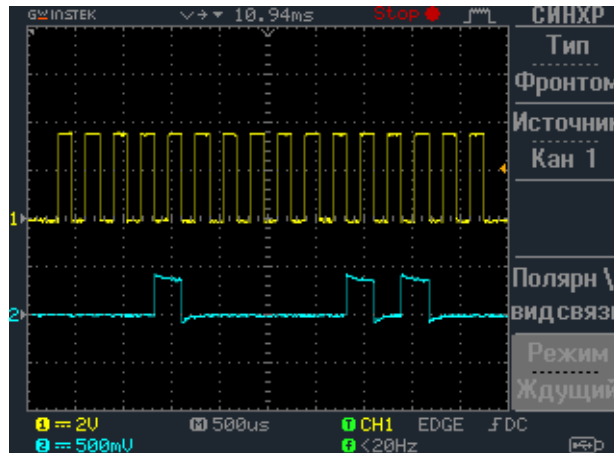


Рисунок 4.4 – Временная диаграммы команд установки адреса столбца

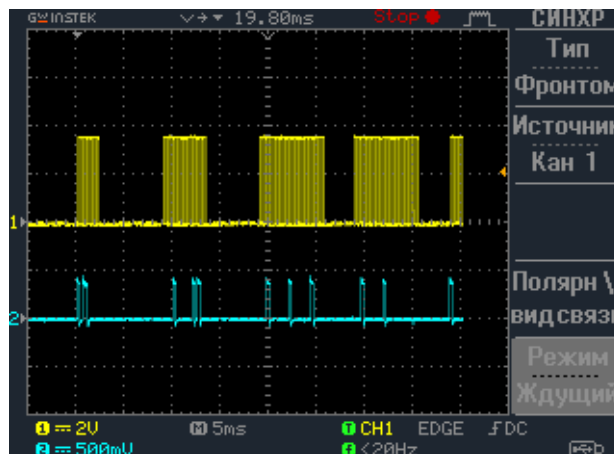


Рисунок 4.5 – Временная диаграммы начала процесса записи данных

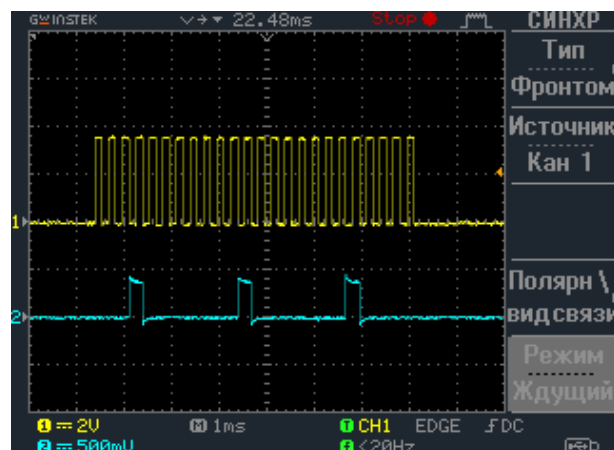


Рисунок 4.6 – Временная диаграммы записи первых трех байт символа

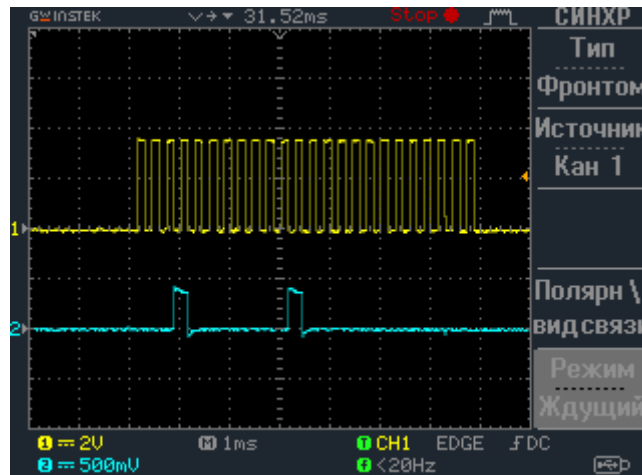


Рисунок 4.7 – Временная диаграммы записи последних трех байт символа

```
//lab 5
#include <msp430.h>
#include <math.h>
#include <stdint.h>

typedef unsigned char uint8_t;

#define SET_COLUMN_ADDRESS_LSB      0x00
#define SET_COLUMN_ADDRESS_MSB     0x10
#define SET_PAGE_ADDRESS            0xB0

#define SET_SEG_DIRECTION           0xA0
#define SET_COM_DIRECTION           0xC0

#define SET_POWER_CONTROL           0x2F
#define SET_SCROLL_LINE             0x40
#define SET_VLCD_RESISTOR_RATIO     0x27
#define SET_ELECTRONIC_VOLUME_MSB   0x81
#define SET_ELECTRONIC_VOLUME_LSB   0x0F
#define SET_ALL_PIXEL_ON            0xA4
#define SET_INVERSE_DISPLAY         0xA6
#define SET_DISPLAY_ENABLE          0xAF
#define SET_LCD_BIAS_RATIO          0xA2
#define SET_ADV_PROGRAM_CONTROL0_MSB 0xFA
#define SET_ADV_PROGRAM_CONTROL0_LSB 0x90

#define NONE                        0
#define READ_X_AXIS_DATA            0x18
#define READ_Y_AXIS_DATA            0x19
#define READ_Z_AXIS_DATA            0x20

// ACCELEROMETER REGISTER DEFINITIONS
#define REVID 0x01
#define CTRL 0x02
#define MODE_400 0x04 // Measurement mode 400 Hz ODR
#define DOUTX 0x06
#define DOUTY 0x07
#define DOUTZ 0x08
```

```

#define G_RANGE_2 0x80 // 2g range
#define I2C_DIS 0x10 // I2C disabled

#define CD BIT6
#define CS BIT4

#define PAGES 12
#define COLUMNS 9

int Mirror_Mode=0;

int MAPPING_VALUES[] = { 4571, 2286, 1141, 571, 286, 143, 71 };
uint8_t BITx[] = { BIT6, BIT5, BIT4, BIT3, BIT2, BIT1, BIT0 };

uint8_t MODE_COMMANDS[2][1] = { {SET_SEG_DIRECTION}, {SET_SEG_DIRECTION | 1} };

uint8_t Dogs102x6_initMacro[] = {
    SET_SCROLL_LINE,
    {SET_SEG_DIRECTION},
    {SET_COM_DIRECTION | 1},
    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
};

int inverted = 0;
int COLUMN_START_ADDRESS = 121;

uint8_t symbols[12][6] = {
    {0xff, 0x81, 0x81, 0x81, 0x81, 0xff}, // UPPER 0 INDEX 0
    {0x08, 0x10, 0x20, 0x40, 0x80, 0xff}, // UPPER 1 INDEX 0
    {0x9f, 0x91, 0x91, 0x91, 0x91, 0xf1}, // UPPER 2 INDEX 0
    {0x91, 0x91, 0x91, 0x91, 0x91, 0xff}, // UPPER 3 INDEX 0
    {0xf0, 0x10, 0x10, 0x10, 0x10, 0xff}, // UPPER 4 INDEX 0
    {0xf1, 0x91, 0x91, 0x91, 0x91, 0x9f}, // UPPER 5 INDEX 0
    {0xff, 0x91, 0x91, 0x91, 0x91, 0x9f}, // UPPER 6 INDEX 0
    {0x80, 0x80, 0x80, 0x80, 0x80, 0xff}, // UPPER 7 INDEX 0

```

```

    {0xff, 0x91, 0x91, 0x91, 0x91, 0xff}, // UPPER 8 INDEX 0

    {0xf1, 0x91, 0x91, 0x91, 0x91, 0xff}, // UPPER 9 INDEX 0

    {0x00, 0x08, 0x08, 0x3e, 0x08, 0x08}, // UPPER + INDEX 0

    {0x00, 0x08, 0x08, 0x08, 0x08, 0x08}, // UPPER - INDEX 0
};

uint8_t CMA3000_writeCommand(uint8_t byte_one, uint8_t byte_two);
void CMA3000_init(void);
int calculateAngleFromProjection(double projection);
long int parseProjectionByte(uint8_t projection_byte);
int8_t Cma3000_readRegister(int8_t Address);

int getNumberLength(long int number);
void printNumber(long int angle);

void Dogs102x6_clearScreen(void);
void Dogs102x6_setAddress(uint8_t pa, uint8_t ca);
void Dogs102x6_writeData(uint8_t* sData, uint8_t i);
void Dogs102x6_writeCommand(uint8_t* sCmd, uint8_t i);
void Dogs102x6_backlightInit(void);
void Dogs102x6_init(void);

#define CHECK_CYCLES 2000
void timer_init(void)
{
    TA0CCR0 = CHECK_CYCLES;
    TA0CTL |= TASSEL__ACLK;
    TA0CTL |= MC__UP;
    TA0CTL |= ID__1;

    TA1CCR0 = CHECK_CYCLES;
    TA1CTL |= TASSEL__ACLK;
    TA1CTL |= MC__UP;
    TA1CTL |= ID__1;
}

void btn_init(void)
{
    P1DIR &= ~BIT7;
    P1OUT |= BIT7;
    P1REN |= BIT7;
    P1IFG &= ~BIT7;
    P1IES |= BIT7;
    P1IE |= BIT7;

    P2DIR &= ~BIT2;
    P2OUT |= BIT2;
    P2REN |= BIT2;
    P2IFG &= ~BIT2;
    P2IES |= BIT2;
    P2IE |= BIT2;
}

```

```

void Dogs102x6_setMirrorDisplay()
{
    uint8_t cmd[] = {SET_COM_DIRECTION};
    if(Mirror_Mode == 1)
    {
        cmd[0] = SET_COM_DIRECTION + 0x08;
    }
    else
    {
        cmd[0] = SET_COM_DIRECTION ;
    }
    Dogs102x6_writeCommand(cmd, 1);
}

void inv_lcd(){

    uint8_t cmd[1];

    if(inverted){
        inverted = 0;
        cmd[0] = SET_INVERSE_DISPLAY;
    }
    else{
        inverted = 1;
        cmd[0] = 0xA7;
    }
    Dogs102x6_writeCommand(cmd, 1);
}

//#pragma vector = PORT1_VECTOR
//__interrupt void HandleS1PreBounce(void) {
//    TA0R = 0;
//    TA0CCTL0 = CCIE;
//
//
//    P1IFG &= ~BIT7;
//}
//
//#pragma vector = PORT2_VECTOR
//__interrupt void HandleS2PreBounce(void) {
//    TA1R = 0;
//    TA1CCTL0 = CCIE;
//
//    // P2IFG &= ~BIT2;
//    P2IFG &= ~BIT2;
//}
//
//#pragma vector = TIMER0_A0_VECTOR
//__interrupt void TimerS1(void) {
//    TA0CCTL0 &= ~CCIE;
//    if (P1IN & BIT7) return;
//
//    //inv_lcd();
//    Mirror_Mode^=1;
//
//    Dogs102x6_clearScreen();
//    Dogs102x6_setMirrorDisplay();

```



```

//    //printNumber(CUR_NUM);
//}
//
//#pragma vector = TIMER1_A0_VECTOR
//__interrupt void TimerS2(void) {
//    TA1CCTL0 &= ~CCIE;
//    //if (P2IN & BIT5) return;
//
//
//    volatile uint8_t xProjectionByte = CMA3000_writeCommand(READ_X_AXIS_DATA, NONE);
//    volatile uint8_t zProjectionByte = CMA3000_writeCommand(READ_Z_AXIS_DATA, NONE);
//
//    volatile long int xAxisProjection = parseProjectionByte(xProjectionByte);
//    volatile long int zAxisProjection = parseProjectionByte(zProjectionByte);
//
//    //Dogs102x6_clearScreen();
//
//    //printNumber(zAxisProjection);
//
//    // ось Z
//
//    uint8_t Cma3000_zAccel = zProjectionByte;
//    Cma3000_zAccel *= 1.57;
//    Cma3000_zAccel = 90 - Cma3000_zAccel;
//    printNumber(Cma3000_zAccel);
//
//    //// // ось X
//    ////
//    //// int angle = calculateAngleFromProjection((double) xAxisProjection);
//    //// angle *= zAxisProjection <= 0 ? 1 : -1;
//    ////
//    //// Dogs102x6_clearScreen();
//    ////     printNumber(angle);
//    //}
//
//#pragma vector = PORT2_VECTOR
//__interrupt void accelerometerInterrupt(void) {
//    volatile uint8_t xProjectionByte = Cma3000_readRegister(DOUTX);
//    volatile uint8_t yProjectionByte = Cma3000_readRegister(DOUTY);
//    volatile uint8_t zProjectionByte = Cma3000_readRegister(DOUTZ);
//
//    volatile long int xAxisProjection = parseProjectionByte(xProjectionByte);
//    volatile long int yAxisProjection = parseProjectionByte(yProjectionByte);
//    volatile long int zAxisProjection = parseProjectionByte(zProjectionByte);
//
//    // Dogs102x6_clearScreen();
//    // printNumber(zAxisProjection);
//
//
//    // ось X
//
//    // int angle = calculateAngleFromProjection((double) xAxisProjection);
//    // angle *= zAxisProjection <= 0 ? 1 : -1;
//    //
//    // Dogs102x6_clearScreen();
//    // printNumber(angle);
//
//    // ось Z

```

```

        int angle = calculateAngleFromProjection((double) zAxisProjection);
        angle *= yAxisProjection <= 0 && xAxisProjection <= 0 ? -1 : +1;

        Dogs102x6_clearScreen();
        printNumber(angle);

        // ось y
        // int angle = calculateAngleFromProjection((double) yAxisProjection);
        // angle *= xAxisProjection <= 0 ? 1 : -1;
        //
        // Dogs102x6_clearScreen();
        // printNumber(angle);

    }

#pragma vector = PORT1_VECTOR
__interrupt void buttonS1(void)
{
    // volatile int i = 0;
    //
    // for (i = 0; i < 2000; i++);

    __delay_cycles(7000);

    if ((P1IN & BIT7) == 0) {

        Mirror_Mode^=1;

        Dogs102x6_clearScreen();
        Dogs102x6_setMirrorDisplay();
        //printNumber();

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

    P1IFG = 0;
}

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;

    // P2DIR &= ~BIT2;
    // P2OUT |= BIT2;
    // P2REN |= BIT2;
    // P2IE |= BIT2;
    // P2IES |= BIT2;
    // P2IFG = 0;
    //
    // P1DIR |= BIT4;
    // P1OUT &= ~BIT4;

    Dogs102x6_init();
    Dogs102x6_backlightInit();
    Dogs102x6_clearScreen();

```

```

    timer_init();
    btn_init();

    CMA3000_init();

    Dogs102x6_clearScreen();
    // printNumber(CUR_NUM);

    __bis_SR_register(LPM0_bits + GIE);

    return 0;
}

void printNumber(long CURRENT_NUMBER) {
    int nDigits = getNumberLength(CURRENT_NUMBER);
    int number = CURRENT_NUMBER;

    int COL_ADR=COLUMN_START_ADDRESS - nDigits*8;

    int pa ;
    if(Mirror_Mode==0)pa=7;
    else pa=0;

    if(number < 0) {
        Dogs102x6_setAddress( pa, COL_ADR);
        Dogs102x6_writeData(symbols[11], 6);
    //     Dogs102x6_setAddress(LOWER_PAGE, COL_ADR);
    //     Dogs102x6_writeData(symbols[22], 6);
    } else {
        Dogs102x6_setAddress( pa, COL_ADR);
        Dogs102x6_writeData(symbols[10], 6);
    //     Dogs102x6_setAddress(UPPER_PAGE, COL_ADR);
    //     Dogs102x6_writeData(symbols[21], 6);
    //     Dogs102x6_setAddress(LOWER_PAGE, COL_ADR);
    //     Dogs102x6_writeData(symbols[20], 6);
    }

    int i = 0;
    int divider = pow(10, nDigits - 1);
    number = abs(number);

    for (i = 1; i <= nDigits; i++) {
        int digit = number / divider;

    //     Dogs102x6_setAddress(UPPER_PAGE, COL_ADR + (i * 2) + (i * 6));
    //     Dogs102x6_writeData(symbols[digit * 2 + 1], 6);
        Dogs102x6_setAddress(pa, COL_ADR + (i * 2) + (i * 6));
        Dogs102x6_writeData(symbols[digit], 6);

        number = number % divider;
        divider /= 10;
    }
}

int getNumberLength(long int number) {
    int length = 0;
    number = fabs1(number);

```

```

        if(number == 0)
            return 1;
        while(number) {
            number /= 10;
            length++;
        }
        return length;
    }

void Dogs102x6_clearScreen(void){
    uint8_t LcdData[] = { 0x00 };
    uint8_t p, c;
    for (p = 0; p < 8; p++){
        Dogs102x6_setAddress(p, 0);
        for (c = 0; c < 132; c++)
            Dogs102x6_writeData(LcdData, 1);
    }
}

void Dogs102x6_setAddress(uint8_t pa, uint8_t ca){
    uint8_t cmd[1];
    // ca-=1;
    if (pa > 7)
        pa = 7;
    if (ca > 131)
        ca = 131;
    cmd[0] = SET_PAGE_ADDRESS + pa;
    // cmd[0] = SET_PAGE_ADDRESS + ( pa );
    uint8_t H = 0x00;
    uint8_t L = 0x00;
    uint8_t ColumnAddress[] = { SET_COLUMN_ADDRESS_MSB, SET_COLUMN_ADDRESS_LSB };
    L = (ca & 0x0F);
    H = (ca & 0xF0);
    H = (H >> 4);
    ColumnAddress[0] = SET_COLUMN_ADDRESS_LSB + L;
    ColumnAddress[1] = SET_COLUMN_ADDRESS_MSB + H;
    Dogs102x6_writeCommand(cmd, 1);
    Dogs102x6_writeCommand(ColumnAddress, 2);
}

void Dogs102x6_writeData(uint8_t* sData, uint8_t i){
    P7OUT &= ~CS;
    P5OUT |= CD;
    while (i){
        while (!(UCB1IFG & UCTXIFG));
        UCB1TXBUF = *sData;
        sData++;
        i--;
    }
    while (UCB1STAT & UCBUSY);
    UCB1RXBUF;
    P7OUT |= CS;
}

void Dogs102x6_writeCommand(uint8_t* sCmd, uint8_t i){
    P7OUT &= ~CS;
    P5OUT &= ~CD;

```

```

        while (i){
            while (!(UCB1IFG & UCTXIFG));
            UCB1TXBUF = *sCmd;
            sCmd++;
            i--;
        }
        while (UCB1STAT & UCBUSY);
        UCB1RXBUF;
        P7OUT |= CS;
    }

void Dogs102x6_backlightInit(void){
    P7DIR |= BIT6;
    P7OUT |= BIT6;
    P7SEL &= ~BIT6;
}

void Dogs102x6_init(void){
    P5DIR |= BIT7;
    P5OUT &= BIT7;
    P5OUT |= BIT7;

    P7DIR |= CS;

    P5DIR |= CD;
    P5OUT &= ~CD;

    P4SEL |= BIT1;
    P4DIR |= BIT1;

    P4SEL |= BIT3;
    P4DIR |= BIT3;

    UCB1CTL1 = UCSSEL_2 + UCSWRST;
    UCB1BR0 = 0x02;
    UCB1BR1 = 0;

    UCB1CTL0 = UCCKPH + UCMSB + UCMST + UCMODE_0 + UCSYNC;

    UCB1CTL1 &= ~UCSWRST;
    UCB1IFG &= ~UCRXIFG;

    Dogs102x6_writeCommand(Dogs102x6_initMacro, 13);
}

void CMA3000_init(void) {
    P2DIR &= ~BIT5;          // mode: input
    P2OUT |= BIT5;
    P2REN |= BIT5;          // enable pull up resistor
    P2IE  |= BIT5;          // interrupt enable
    P2IES &= ~BIT5;         // process on interrupt's front
    P2IFG &= ~BIT5;         // clear interrupt flag

    // set up cma3000 (CBS - Chip Select (active - 0))
    P3DIR |= BIT5;          // mode: output
    P3OUT |= BIT5;          // disable cma3000 SPI data transfer

```

```

// set up ACCEL_SCK (SCK - Serial Clock)
P2DIR |= BIT7;          // mode: output
P2SEL |= BIT7;          // clk is UCA0CLK

// Setup SPI communication
P3DIR |= (BIT3 | BIT6);  // Set MOSI and PWM pins to output mode
P3DIR &= ~BIT4;          // Set MISO to input mode
P3SEL |= (BIT3 | BIT4);  // Set mode : P3.3 - UCA0SIMO , P3.4 - UCA0SOMI
P3OUT |= BIT6;           // Power cma3000
UCA0CTL1 = UCSSEL_2 | UCSWRST;

UCA0BR0 = 0x30;
UCA0BR1 = 0x0;

UCA0CTL0 = UCCKPH & ~UCCKPL | UCMSB | UCMST | UCSYNC | UCMODE_0;

UCA0CTL1 &= ~UCSWRST;

// dummy read from REVID
CMA3000_writeCommand(0x04, NONE);
__delay_cycles(1250);

// write to CTRL register
CMA3000_writeCommand(0x0A, BIT4 | BIT2);
__delay_cycles(25000);

// Activate measurement mode: 2g/400Hz
CMA3000_writeCommand(CTRL, G_RANGE_2 | I2C_DIS | MODE_400);

// Settling time per DS = 10ms
// __delay_cycles(1000 * TICKSPERUS);
__delay_cycles(25000);
}

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;
    }

    long int parseProjectionByte(uint8_t projectionByte) {
        int i = 0;
        long 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;
    }

    int calculateAngleFromProjection(double projection) {
        projection /= 1000;
        projection = projection > 1 ? 1 : projection < -1 ? -1 : projection;

        double angle = acos(projection);
        angle *= 57.3;

        return (int) angle;
    }

    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
        P3OUT &= ~BIT5;

        // 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
P3OUT |= BIT5;

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

```

ВЫВОД

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, выводящая данные акселерометра на ЖКИ и проанализированы диаграмма работы интерфейса SPI ЖКИ