БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 5
Тема: «Последовательный интерфейс SPI ЖКИ. Акселерометр»
Вариант №8

Выполнил:
студент группы 150503 Ходосевич М.А.

Проверил:
ассистент каф. ЭВМ _____ Шеменков В.В.

Минск
2024

1. Постановка задачи

Изучить принципы организации последовательного интерфейса SPI и подключения устройств на его основе на базе микроконтроллера MSP430F5529.

В соответствии с вариантом написать программу, которая получает данные от акселерометра и в требуемом виде отражает их на экране. По нажатию кнопки S1 зеркально отразить результат, используя команды для ЖКИ. Снять временные диаграммы всех линий интерфейса SPI (USCI_B1). Задания варианта 8: место индикации – правый нижний угол; ориентация текста – 180º; зеркальное отражение – вертикально, весь экран; ось измерений акселерометра – X.

2. Теоретические сведения

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 рис. 2.1. 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.

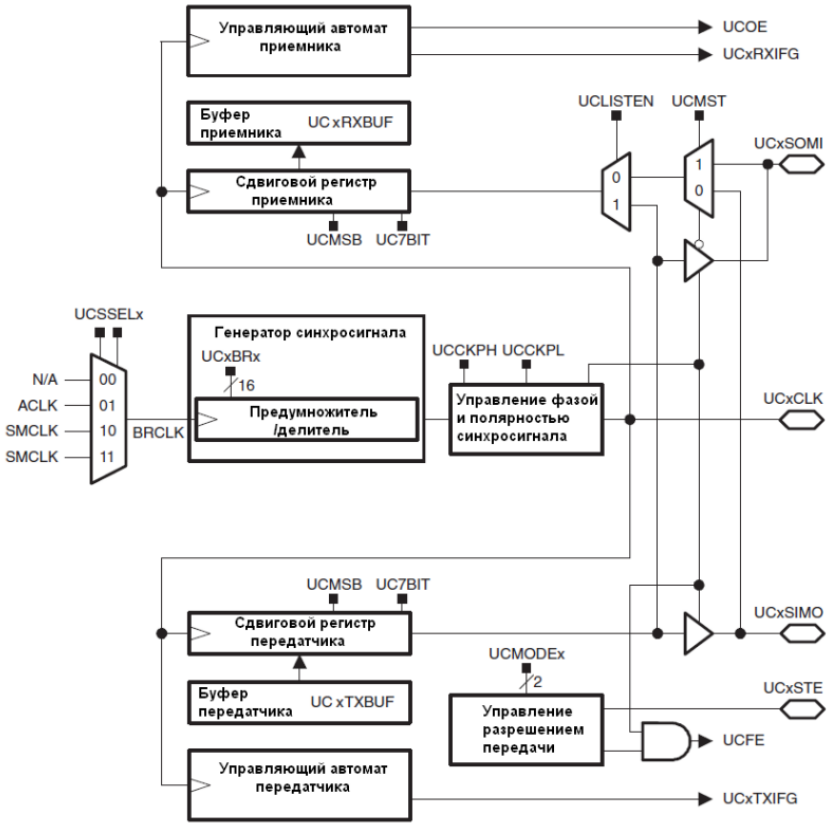

Рисунок 2.1 – Структура интерфейса SPI

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 Masterустройства 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 Slaveустройства тактовый генератор отключен, а передача начинается с выставлением тактового сигнала Master-устройством. Наличие передачи определяется флагом UCBUSY = 1.

На рисунке 2.2 в таблице приведены 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

Рисунок 2.2 – Таблица регистров интерфейса

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.

На рисунке 2.3 – приведена таблица соответствия выводов устройства выводам микроконтроллера MSP430F5529 и их назначение.

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

Рисунок 2.3 – Таблица соответствия выводов ЖКИ экрана

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.

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 5.5.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.

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. Это приведено на рисунке 2.4.

Рисунок 2.4 – 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 пискселей (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 5.7. Контроллер ЖКИ при формировании сигнала сброса требует ожидания 5- 10 мс, при включении питания ожидания не требуется.

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.

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 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.

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.

На рисунке 2.5 приведена типичная последовательность действий при инициализации акселерометра:

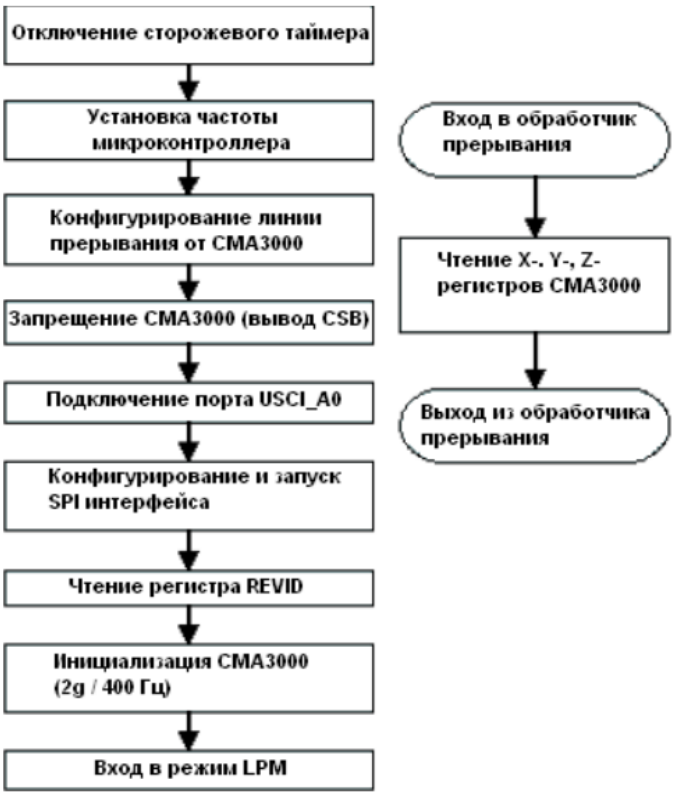

Рисунок 2.5 – Типичная последовательность при инициализации CMA3000-D01

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.

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. Уровень сигнала на линии можно измерить, подключив щупы осциллографа к соответствующему выводу разъема и GND разъема J5.

3. Выполнение работы

Написанная программа получает данные от акселерометра и отражает их на экране в правом нижнем углу при ориентации текста на 180º. По нажатию кнопки S1 вертикально зеркально отражается весь экран.

На рисунке 3.1 показан сигнал SIMO с вывода P4.1 линии интерфейса SPI (USCI_B1).

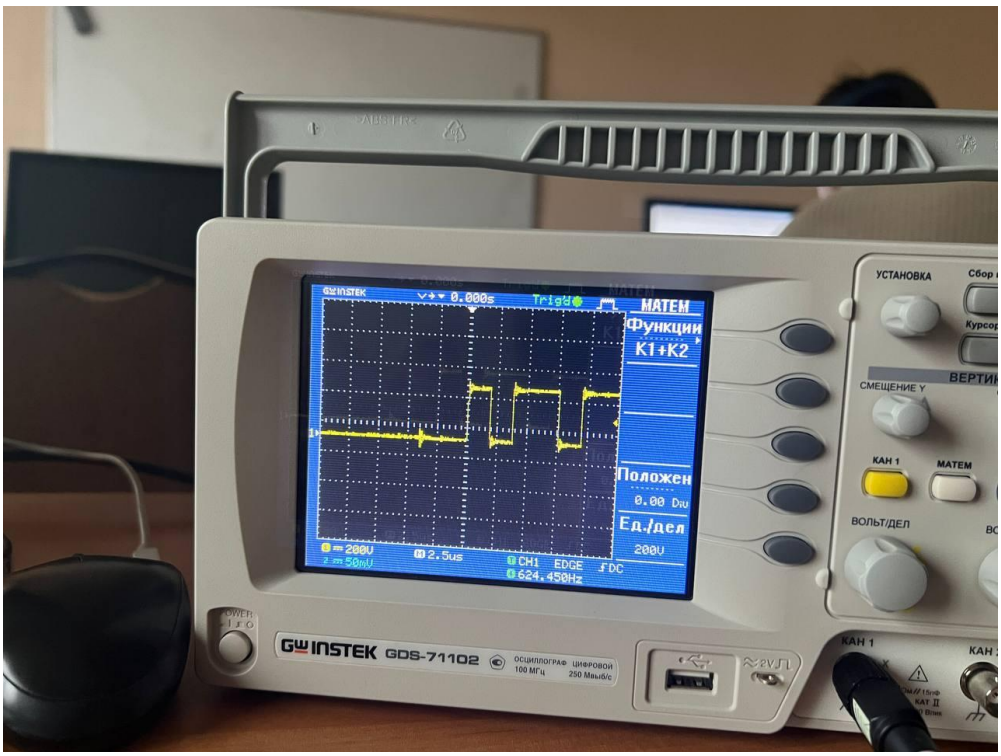

Рисунок 3.1 – Временная диаграмма вывода P4.1

На рисунке 3.2 показан сигнал SOMI с вывода P4.2 линии интерфейса SPI (USCI_B1).

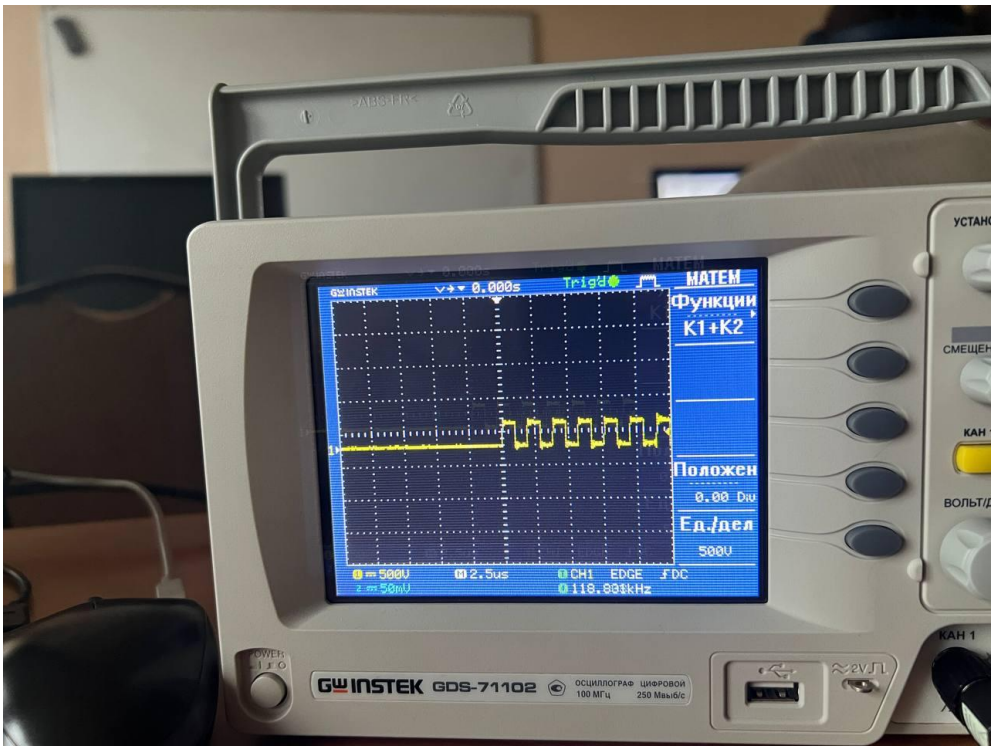

Рисунок 3.2 – Временная диаграмма вывода P4.2

На рисунке 3.3 показан сигнал CLK с вывода P4.3 линии интерфейса SPI (USCI_B1).

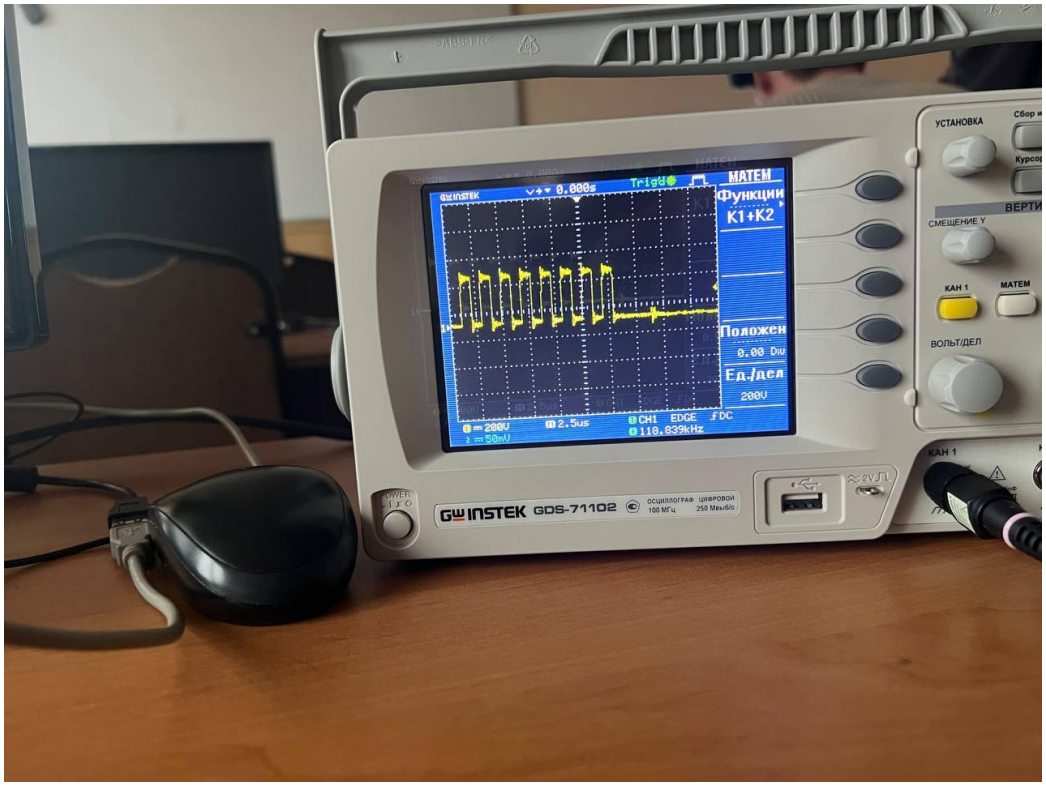

Рисунок 3.3 – Временная диаграмма вывода P4.3

4. Листинг программы

```
#include <msp430.h>

// Определения для команд управления ЖК-дисплеем и данных акселерометра
#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

// Определения регистров акселерометра
#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

// Инициализирует ориентацию сегмента ЖК-дисплея для зеркального
отображения
void Dogs102x6_setMirrorSegDisplay();
// Инициализирует ориентацию столбцов ЖК-дисплея для зеркального
отображения
void Dogs102x6_setMirrorColDisplay();
int MirrorColMode=0;
int MirrorSegMode=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,
     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 = 100;
int PAGE_START_ADDRESS = 7;

// Символы для чисел и знаков, отображаемых на ЖК-дисплее
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
};

// Записывает данные в указанный регистр акселерометра
uint8_t CMA3000_writeCommand(uint8_t byte_one, uint8_t byte_two);
// Инициализирует конфигурацию акселерометра
void CMA3000_init(void);
// Вычисляет угол по заданному значению проекции
int calculateAngleFromProjection(double projection);
// Преобразует байт проекции акселерометра в целое число
long int parseProjectionByte(uint8_t projection_byte);
// Считывает регистр с акселерометра и возвращает результат
int8_t Cma3000_readRegister(int8_t Address);

// Вычисляет количество цифр в числе
int getNumberLength(long int number);
// Отображает числовое значение на ЖК-экране
void printNumber(long int angle);

// Очищает весь ЖК-экран
void Dogs102x6_clearScreen(void);
// Устанавливает адрес ЖК-дисплея для записи
void Dogs102x6_setAddress(uint8_t pa, uint8_t ca);
//записывает данные на ЖК-дисплей
void Dogs102x6_writeData(uint8_t* sData, uint8_t i);
//пишет команду для ЖК-дисплея
void Dogs102x6_writeCommand(uint8_t* sCmd, uint8_t i);
//инициализирует подсветку
void Dogs102x6_backlightInit(void);
//инициализирует настройки и конфигурацию ЛСД
void Dogs102x6_init(void);

#define CHECK_CYCLES 2000

// Инициализирует таймер для отсчета времени события и обновления экрана
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;
}

// Инициализирует кнопки и их поведение прерывания
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;
    }

    // Устанавливает режим зеркального отображения направления столбцов на
ЖК-дисплее
    void Dogs102x6_setMirrorColDisplay()
    {
        uint8_t cmd[] = {SET_COM_DIRECTION};

        if(MirrorColMode == 1)
        {
            cmd[0] = SET_COM_DIRECTION + 0x08;
        }
        else
        {
            cmd[0] = SET_COM_DIRECTION ;
        }
        Dogs102x6_writeCommand(cmd, 1);
    }

    // Устанавливает режим зеркального отображения направления сегмента на
ЖК-дисплее
    void Dogs102x6_setMirrorSegDisplay()
    {
        uint8_t cmd[] = {SET_SEG_DIRECTION};

        if(MirrorSegMode == 1)
        {
            cmd[0] = SET_SEG_DIRECTION + 0x01;
        }
        else
        {
            cmd[0] = SET_SEG_DIRECTION ;
        }
        Dogs102x6_writeCommand(cmd, 1);
    }

    // Переключает режим инверсии для ЖК-дисплея
    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);
    }

    // Обработчик прерываний для обработки данных акселерометра
    #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);
            // ось x
            int angle = calculateAngleFromProjection((double) xAxisProjection);
            angle *= zAxisProjection <= 0 ? 1 : -1;
            Dogs102x6_clearScreen();
            printNumber(angle);
      }

      // Обработчик прерываний для кнопки S1 для переключения зеркалирования
сегмента
      #pragma vector = PORT1_VECTOR
      __interrupt void buttonS1(void)
      {
            __delay_cycles(7000);
            if ((P1IN & BIT7) == 0) {
                MirrorSegMode^=1;
                if(MirrorSegMode == 0)COLUMN_START_ADDRESS = 30 ;
                else
                    COLUMN_START_ADDRESS = 0 ;
                Dogs102x6_clearScreen();
                Dogs102x6_setMirrorSegDisplay();
            }
            P1IFG = 0;
      }

      int getNumberLength(long int number) {
            int length = 0;
            number = abs(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 | 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) {
     volatile 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) {
     long int projectionValue = 0;
     int i = 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;
       }

      void printNumber(long CURRENT_NUMBER) {
            int nDigits = getNumberLength(CURRENT_NUMBER);
            int number = CURRENT_NUMBER;
            int COL_ADR=COLUMN_START_ADDRESS;
            int pa = PAGE_START_ADDRESS;
            if(number < 0) {
                Dogs102x6_setAddress( pa, COL_ADR);
                Dogs102x6_writeData(symbols[11], 6);
            } else {
```

```
            Dogs102x6_setAddress( pa, COL_ADR);
            Dogs102x6_writeData(symbols[10], 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(pa, COL_ADR + (i * 2) + (i * 6));
            Dogs102x6_writeData(symbols[digit], 6);
            number = number % divider;
            divider /= 10;
        }
}

// Основной цикл инициализации и выполнения программы
int main(void) {
        WDTCTL = WDTPW | WDTHOLD;
        Dogs102x6_init();
        Dogs102x6_backlightInit();
        Dogs102x6_clearScreen();
        timer_init();
        btn_init();
        CMA3000_init();
        MirrorColMode=0;
        Dogs102x6_setMirrorColDisplay();
        MirrorSegMode=0;
        Dogs102x6_setMirrorSegDisplay();
        COLUMN_START_ADDRESS = 30 ;
        Dogs102x6_clearScreen();
        __bis_SR_register(LPM0_bits + GIE);
        return 0;
}
```

5. Заключение

В ходе лабораторной работы были изучены принципы организации последовательного интерфейса SPI и подключения устройств на его основе на базе микроконтроллера MSP430F5529. Удалось написать программу с использованием интерфейса SPI, ЖКИ и акселерометра в соответствии с заданием. Программа получает данные от акселерометра и в требуемом виде (в соответствие с вариантом) отражает их на экране.