Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет по лабораторной работе № 5

Вариант № 3

Проверил: Выполнили:

Шеменков В.В. ст. гр. №050504 Рубель А.А.

Чеботарев В.С.

Минск 2023

1. Постановка задачи

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

1. теоретические сведения

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.

1. 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.

ЖКИ экран DOGS102W-6 поддерживает разрешение 102 х 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.

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).

1. Выполнение работы

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. 0xC8. На рисунке 4.2 показана команда установки зеркального порядка строк: На рисунке 4.3 показана команда установки адреса страницы 0: 0xB0.

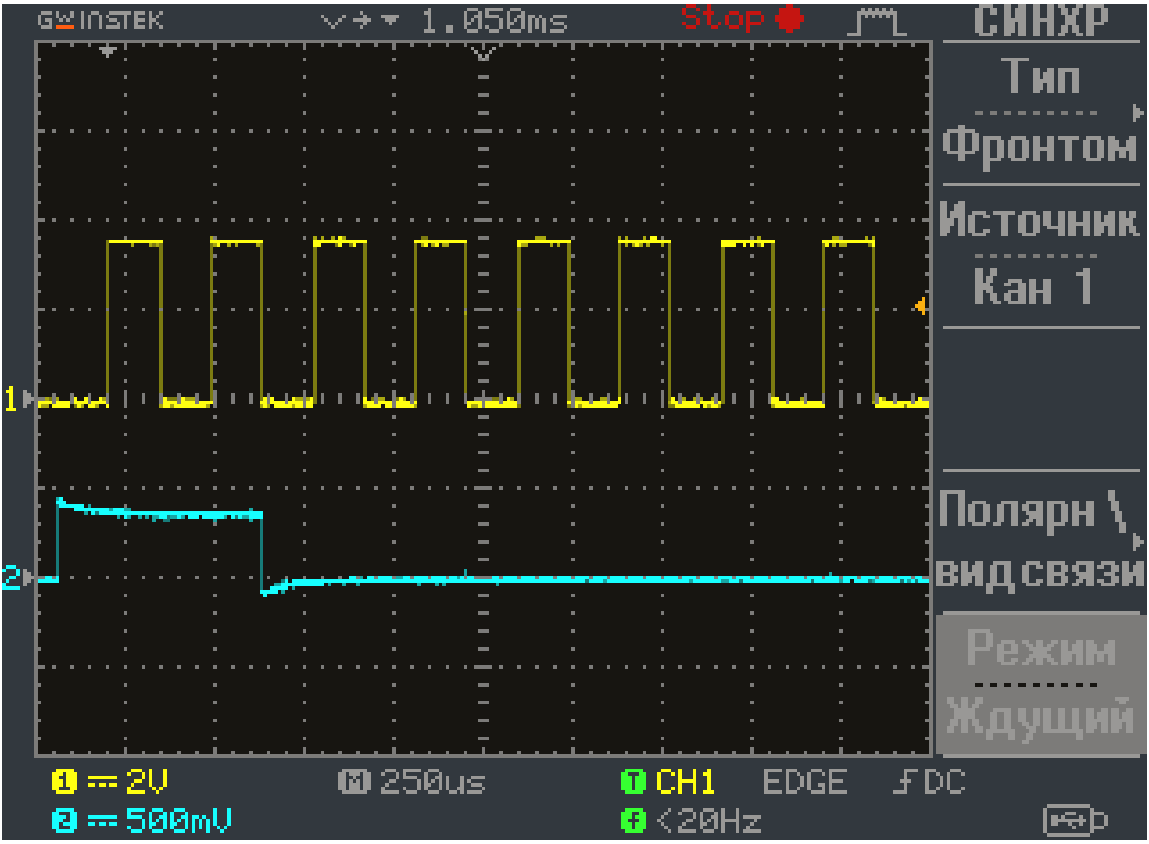


Рисунок 4.1 – Временная диаграммы команды установки нормального порядка строк

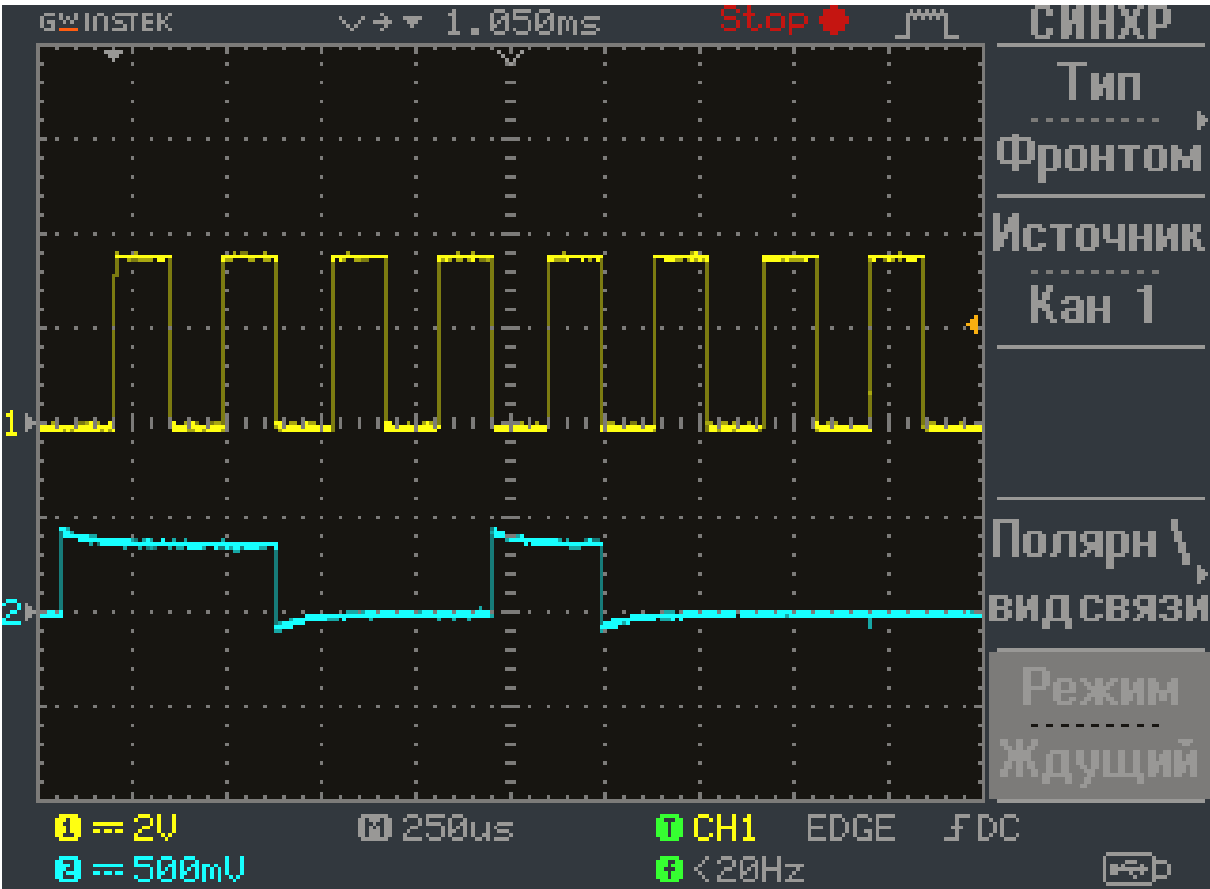


Рисунок 4.2 – Временная диаграммы команды установки зеркального порядка строк

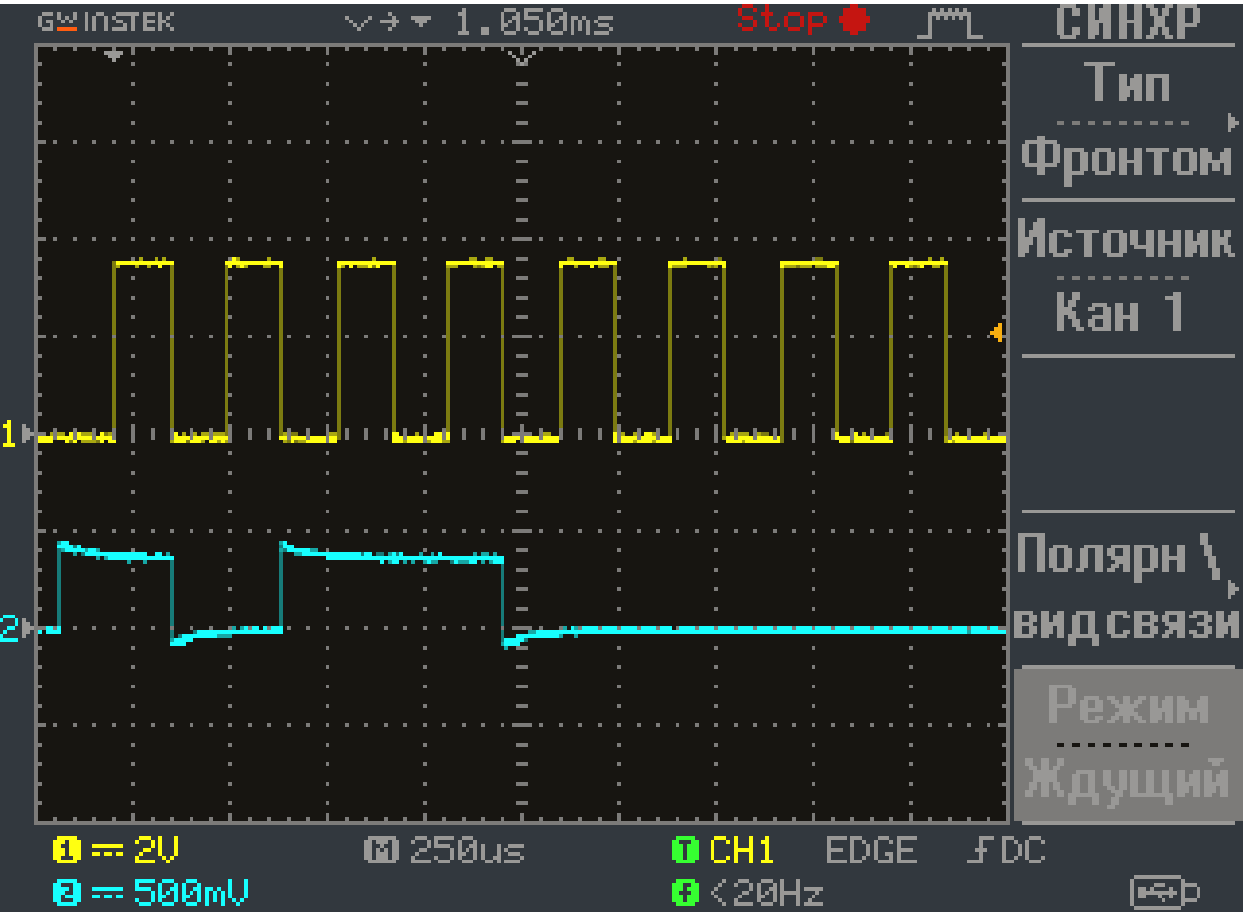


Рисунок 4.3 – Временная диаграммы команды установки страницы

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

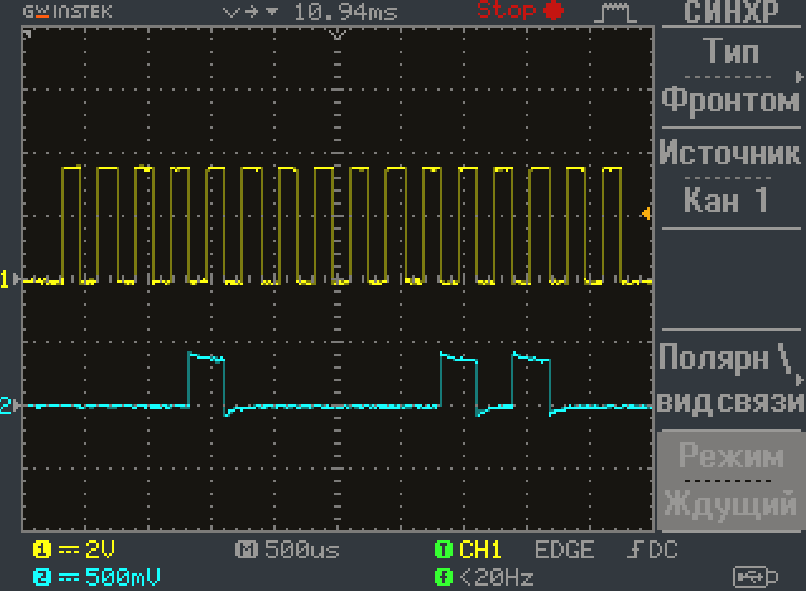


Рисунок 4.4 – Временная диаграммы команд установки адреса столбца

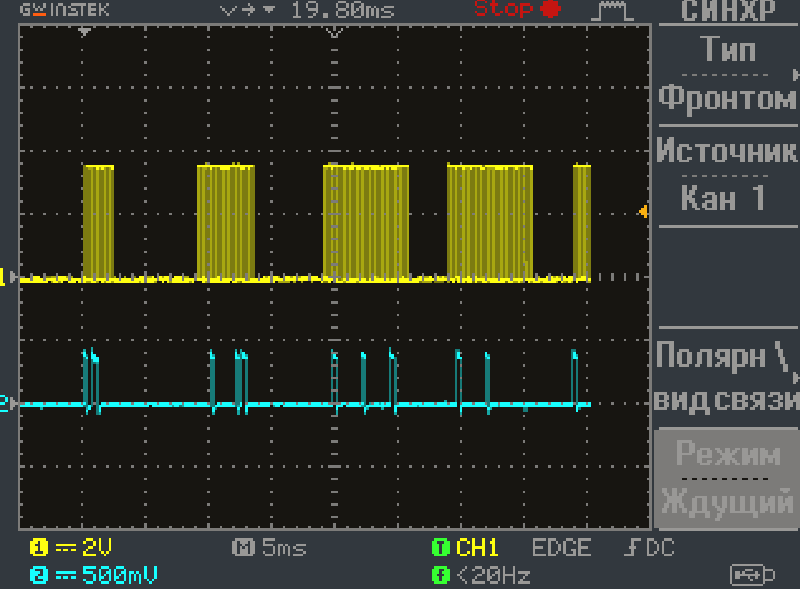


Рисунок 4.5 – Временная диаграммы начала процесса записи данных

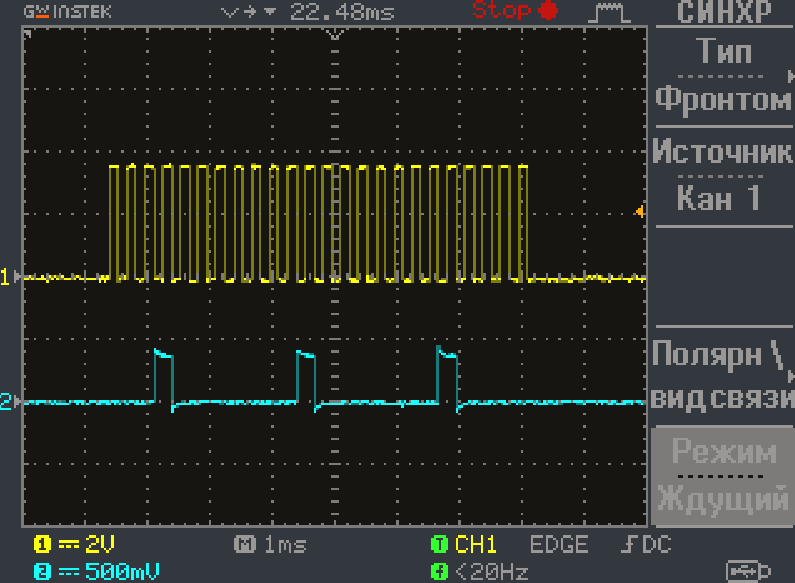


Рисунок 4.6 – Временная диаграммы записи первых трех байт символа

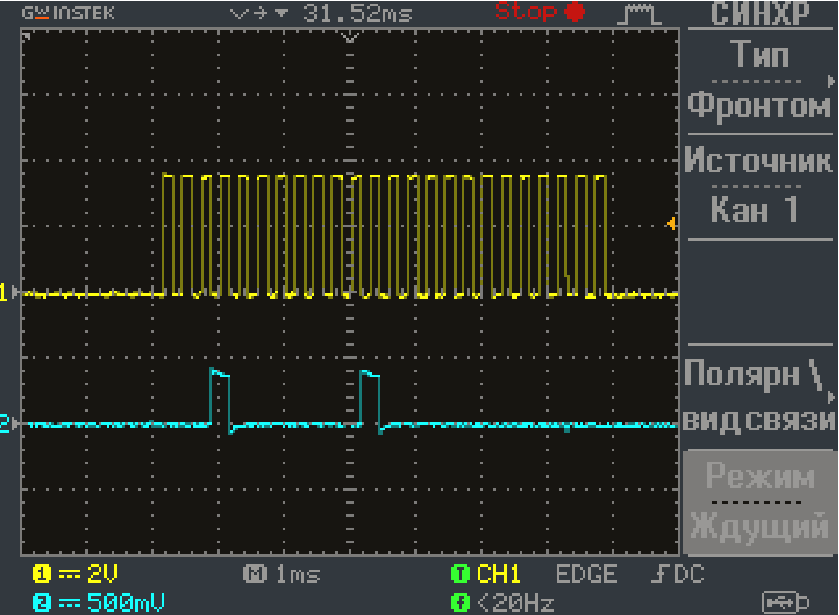


Рисунок 4.7 – Временная диаграммы записи последних трех байт символа

1. листинг кода

#include <msp430.h>

#include <stdint.h>

#include “allDefines.h”

//#include <math.h>

// CONSTANTS

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, 0xB9, 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(uint8\_t dir)

{

uint8\_t cmd;

if (dir == 1) {

cmd = SET\_COM\_DIRECTION\_1;

} else {

cmd = SET\_COM\_DIRECTION;

}

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\_y = parseProjectionByte(Cma3000\_yAccel);

ShowNumber(num\_y);

}

#pragma vector=WDT\_VECTOR

\_\_interrupt void WDTINT()

{

SFRIE1 &=~ WDTIE;

Cma3000\_readAccel();

SFRIE1 |= WDTIE;

}

int invFlg = 0;

#pragma vector = PORT2\_VECTOR

\_\_interrupt void buttonS1(void)

{

volatile int i = 0;

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

if ((P2IN & BIT2) == 0) {

if (invFlg == 0) {

invFlg = 1;

Dogs102x6\_setInverseDisplay(1);

}

else {

invFlg = 0;

Dogs102x6\_setInverseDisplay(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;

}

int fabs\_s(int num) {

if(num > 0){

return num;

}

else{

return num\*(-1);

}

}

//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);

// }

// }

// }

//}

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, 102 - (length\*colSize-j\*colSize), 48 + digit, 0);

}

}

if (num < 0)

{

Dogs102x6\_charDraw(7, 102-(length\*colSize) - colSize, 45, 0);

}

}

else{

if (num < 0)

{

Dogs102x6\_charDraw(7, 102-(length\*colSize) - colSize, 45,0);

}

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), 48 + digit,0);

}

}

}

}

void SetupButtons()

{

P4SEL |= BIT2;

P4DIR |= BIT2;

P4SEL |= BIT1;

P4DIR |= BIT1;

P4SEL |= BIT0;

P4DIR |= BIT0;

P4SEL |= BIT3;

P4DIR |= BIT3;

P3DIR |= BIT7;

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;

}

1. Заключение

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 роанализированы диаграмма работы интерфейса SPI ЖКИ