

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

ІКНІ
Кафедра ПЗ

ЗВІТ

до лабораторної роботи № 4, 5

з дисципліни: *“Основи програмування вбудованих систем”*

на тему: *“Дослідження протоколу SPI на прикладі акселерометра у складі
STM32F4DISCOVERY та LCD-дисплею”*

Варіант 1

Лектор:
доц. каф. ПЗ
Марусенкова Т. А.

Виконав:
ст. гр. ПЗ-32
Бурець В.В.

Прийняв:
доц. каф. ПЗ
Крук О.Г.

« ____ » _____ 2021 р.

Σ= ____

Тема роботи: Дослідження протоколу SPI на прикладі акселерометра у складі STM32F4DISCOVERY та LCD-дисплею.

Мета роботи: Засвоїти принципи роботи інтерфейсу SPI та здобути практичні навички організації взаємодії мікроконтролера з периферійними пристроями через SPI за допомогою бібліотек CMSIS, SPL та HAL, акселерометра LIS302DL у складі оціночної плати stm32f4Discovery та дисплею моделі ET-NOKIA LCD 5110.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Яким є інтерфейс SPI: послідовним чи паралельним?

Інтерфейс SPI є послідовний синхронний повнодуплексний інтерфейс. Тобто вказує на можливість передачі даних одночасно у двох напрямках — від ведучого до веденого і навпаки. Утім, можна налаштувати SPI і на напівдуплексний режим (при цьому можна використовувати на одну лінію менше).

ЗАВДАННЯ

1. Написати функцію ініціалізації GPIO та SPI для обміну даними з акселерометром за допомогою CMSIS SPI Driver.
2. Розробити функції для обміну даними (читання та запису) за допомогою CMSIS SPI Driver.
3. Зчитати модель акселерометра. За допомогою вікна Watch у середовищі Keil uVision перевірити покази акселерометра.
4. Додати у проект функції для ініціалізації дисплею ET-NOKIA LCD 5110. Визначити за наданими кодами, які з виводів GPIO задіяні для обміну даними з дисплеєм і за що відповідає кожен вивід.
5. Підключити макетну плату з дисплеєм до визначених виводів.
6. Написати додаткові функції для роботи з дисплеєм у відповідності до індивідуального завдання.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Варіант 1

1. Написати функцію для очищення дисплею.

ХІД РОБОТИ

Для реалізації завдання я виконав наступні дії:

1. Сконфігурував порти GPIOA та порт GPIOE — для роботи зі акселерометром. Відповідні налаштування реалізував окремими функціями у допоміжному файлі Init.c.

2. У файлі `sri_func.c` реалізував функції для читання та запису в SPI. А також було написано функцію для роботи із акселерометром

Для використання CMSIS-SPI Driver вибрав такі налаштування проекту:

Manage Run-Time Environment				
Software Component	Sel.	Variant	Version	Description
SAI (API)			1.2.0	SAI Driver API for Cortex-M
SPI (API)			2.3.0	SPI Driver API for Cortex-M
Custom	<input type="checkbox"/>		1.0.0	Access to #include Driver_SPI.h file and code template for custom implemer
Multi-Slave	<input type="checkbox"/>		1.0.1	SPI Master to Multi-Slave Driver Wrapper
SPI	<input checked="" type="checkbox"/>		2.15	SPI Driver for STM32F4 Series
USART (API)			2.4.0	USART Driver API for Cortex-M
USB Device (API)			2.3.0	USB Device Driver API for Cortex-M
USB Host (API)			2.3.0	USB Host Driver API for Cortex-M
VIO (API)			0.1.0	Virtual I/O
WiFi (API)			1.1.0	WiFi driver
CMSIS RTOS Validation				CMSIS-RTOS Validation Suite
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
Startup	<input checked="" type="checkbox"/>		2.6.3	System Startup for STMicroelectronics STM32F4 Series
STM32Cube Framework (API)			1.0.0	STM32Cube Framework
Classic	<input checked="" type="checkbox"/>		1.7.9	Configuration via RTE_Device.h
STM32CubeMX	<input type="checkbox"/>		1.0.0	Configuration via STM32CubeMX
STM32Cube HAL				STM32F4xx Hardware Abstraction Layer (HAL) Drivers
ADC	<input type="checkbox"/>		1.7.9	Analog-to-digital converter (ADC) HAL driver
CAN	<input type="checkbox"/>		1.7.9	Controller area network (CAN) HAL driver
CRC	<input type="checkbox"/>		1.7.9	CRC calculation unit (CRC) HAL driver
Common	<input checked="" type="checkbox"/>		1.7.9	Common HAL driver
Validation Output		Description		

Рис. 1. Компоненти, необхідні для роботи з операційною системою CMSIS-SPI Driver

Software Component	Sel.	Variant	Version	Description
EXTI	<input type="checkbox"/>		1.7.9	External interrupts and events (EXTI) controller
Flash	<input type="checkbox"/>		1.7.9	Embedded Flash memory HAL driver
GPIO	<input checked="" type="checkbox"/>		1.7.9	General-purpose I/O (GPIO) HAL driver
HCD	<input type="checkbox"/>		1.7.9	USB Host controller (HCD) HAL driver
I2C	<input type="checkbox"/>		1.7.9	Inter-integrated circuit (I2C) interface HAL driver
I2S	<input type="checkbox"/>		1.7.9	I2S HAL driver
IrDA	<input type="checkbox"/>		1.7.9	IrDA HAL driver
IWDG	<input type="checkbox"/>		1.7.9	Independent watchdog (IWDG) HAL driver
MMC	<input type="checkbox"/>		1.7.9	Multi Media Card (MMC) interface HAL driver
NAND	<input type="checkbox"/>		1.7.9	NAND Flash controller HAL driver
NOR	<input type="checkbox"/>		1.7.9	NOR Flash controller HAL driver
PC Card	<input type="checkbox"/>		1.7.9	PC Card controller HAL driver
PCD	<input type="checkbox"/>		1.7.9	USB Peripheral controller (PCD) HAL driver
PWR	<input checked="" type="checkbox"/>		1.7.9	Power controller (PWR) HAL driver
RCC	<input checked="" type="checkbox"/>		1.7.9	Reset and clock control (RCC) HAL driver
RNG	<input type="checkbox"/>		1.7.9	Random number generator (RNG) HAL driver
RTC	<input type="checkbox"/>		1.7.9	Real-time clock (RTC) HAL driver
SD	<input type="checkbox"/>		1.7.9	Secure digital (SD) interface HAL driver
SMBUS	<input type="checkbox"/>		1.7.9	SMBus (SMBUS) interface HAL driver
SPI	<input checked="" type="checkbox"/>		1.7.9	Serial peripheral interface (SPI) HAL driver
SRAM	<input type="checkbox"/>		1.7.9	SRAM controller (SRAM) HAL driver
Smartcard	<input type="checkbox"/>		1.7.9	Smartcard HAL driver

Рис. 2. Компоненти, необхідні для роботи з операційною системою CMSIS-SPI Driver

КОД ПРОГРАМИ

1) завдання виконане з використанням бібліотеки CMSIS-SPI Driver

Main.c

```
#include <Driver_SPI.h>
#include <string.h>
#include "init.h"
#include "spi_func.h"
#include "stdio.h"

int main(void)
{
    init_gpio();
    init_spi();

    LIS_Axes axes;
    while(1)
    {
        who_am_i();

        //return pointer on array
        memset(&axes, 0, sizeof(LIS_Axes));

        LIS_Read_Axes(&axes);
    }
}
```

init.h

```
#include <stm32f407xx.h>
```

```

#include <stdlib.h>
#include "init.h"

extern ARM_DRIVER_SPI Driver_SPI1;

void init_gpio(void)
{
    //включення тактування периферійного блоку
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOEEN;

    // В становлення пінів для задання режимів роботи
    GPIOA->MODER |= GPIO_MODER_MODER5_1 | GPIO_MODER_MODER6_1 |
    GPIO_MODER_MODER7_1;

    //задається швидкість роботи порта
    GPIOA->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR5_1 | GPIO_OSPEEDER_OSPEEDR6_1 |
    GPIO_OSPEEDER_OSPEEDR7_1;
    GPIOA->PUPDR |= GPIO_PUPDR_PUPD5_1 | GPIO_PUPDR_PUPD6_1 | GPIO_PUPDR_PUPD7_1;

    GPIOE->MODER |= GPIO_MODER_MODER3_0;
    GPIOE->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR3_1;
    GPIOE->PUPDR |= GPIO_PUPDR_PUPD3_1;
}

short init_spi()
{
    RCC->AHB2ENR |= RCC_APB2ENR_SPI1EN;

    //Створення структури ARM драйвер SPI
    ARM_DRIVER_SPI *SPIDrv = &Driver_SPI1;

    SPIDrv->Initialize(NULL);

    SPIDrv->PowerControl(ARM_POWER_FULL);

    //Налаштування модуля SPI, встановлення
    //пристрій є ведучим чи веденим(тут ведений)
    //режим передачі даних повнодуплексний або напівдуплексний(встановлено високий
    //логічний рівень по задньому фронту)
    //встановлення біту з MSB до LSB
    //встановлення довжини даних
    SPIDrv->Control(
        ARM_SPI_MODE_MASTER
        | ARM_SPI_CPOL1_CPHA1
        | ARM_SPI_MSB_LSB
        | ARM_SPI_SS_MASTER_UNUSED
        | ARM_SPI_DATA_BITS(8),
        10000);

    return 0;
}

```

init.h

```

#ifndef INIT_GPIO
#define INIT_GPIO

#include <Driver_SPI.h>
extern ARM_DRIVER_SPI Driver_SPI1;

```

```

void init_gpio(void);
short init_spi(void);

#endif

```

LEDInit.h

```

#ifndef INIT_H
#define INIT_H

void ConfigGPIO(void);
void ConfigEXTI(void);

#endif

```

Spi_func.c

```

#include <Driver_SPI.h>
#include <stm32f407xx.h>
#include "init.h"
#include "spi_func.h"

typedef enum
{
    OFF,
    ON
} CS_Status;

static void CS_OnOff(CS_Status stat)
{
    switch(stat)
    {
        case ON:
            GPIOE->ODR &= ~GPIO_ODR_ODR_3;
            break;
        case OFF:
            GPIOE->ODR |= GPIO_ODR_ODR_3;
            break;
    }
}

int32_t SPI_Read(uint16_t reg, int16_t *val)
{
    ARM_DRIVER_SPI *SPIDrv = &Driver_SPI1;

    //встановлення найстаршого біта для передачі даних
    //setting the unit in the high bit
    CS_OnOff(ON);
    reg |= 0x80;

    //Вказівка із яким регістром працювати
    //send data witch register to use
    SPIDrv->Send(&reg, 1);
}

```

```

//Перевірка чи регістр використовується
//check that the register WHO_AM_I respond
//check the register is busy
while(SPIdrv->GetStatus().busy);

//зчитування даних
//read data
SPIdrv->Receive(val, 1);
while(SPIdrv->GetStatus().busy);

//зняття найстаршого біта для зупинення передачі даних

//unsetting the unit in the high bit
CS_OnOff(OFF);
return 0;
}

int32_t SPI_Write(uint16_t reg, int16_t val)
{
    ARM_DRIVER_SPI *SPIdrv = &Driver_SPI1;

    int16_t data[2];
    data[0] = reg;
    data[1] = val;

    CS_OnOff(ON);

//запис значення акселерометра в регістр
SPIdrv->Send(data, 2);
while(SPIdrv->GetStatus().busy);

if(SPIdrv->GetDataCount() != 2)
    return -1;
CS_OnOff(OFF);

return 0;
}

void LIS_Read_Axes(LIS_Axes *axes)
{
    int16_t data[6] = {0};

    for(uint8_t i = 0; i < 6; ++i)
    {
        //SPI_Write(LIS3DSH_OUT_X_L + i, i);

//зчитування значення показників акселерометра
        SPI_Read(LIS3DSH_OUT_X_L + i, &data[i]);
    }

//запис значення акселерометра в поля структури _LIS_Axes
    axes->X = (int16_t)(((data[1] << 8) + data[0]) * 0.06);
    axes->Y = (int16_t)(((data[3] << 8) + data[2]) * 0.06);
    axes->Z = (int16_t)(((data[5] << 8) + data[4]) * 0.06);
}

int16_t who_am_i(void)
{
    int16_t val = 0;

```

```
//зчитування значення регістра WHO_AM_I
    SPI_Read(WHO_AM_I, &val);
    return val;
}
```

spi_func.h

```
#ifndef SPI_RW
#define SPI_RW

#include <stdint.h>

#define LIS3DSH_OUT_X_L      0x28
#define LIS3DSH_OUT_X_H      0x29
#define LIS3DSH_OUT_Y_L      0x2A
#define LIS3DSH_OUT_Y_H      0x2B
#define LIS3DSH_OUT_Z_L      0x2C
#define LIS3DSH_OUT_Z_H      0x2D

#define WHO_AM_I              0x0F

typedef struct _LIS_Axes
{
    int16_t X;
    int16_t Y;
    int16_t Z;
} LIS_Axes;

int32_t SPI_Read(uint16_t reg, int16_t *val);
int32_t SPI_Write(uint16_t reg, int16_t val);

//func for accelerometer
void LIS_Read_Axes(LIS_Axes *axes);
int16_t who_am_i(void);

#endif
```

2) завдання виконане з використанням бібліотеки CMSIS-SPI Driver та написано функцію для очистки дисплею згідно індивідуального завдання.

Main.c

```
#include <stm32f407xx.h>
#include "main.h"
#include "InitGPIO.h"
#include "LCDInit.h"

static volatile uint16_t delay_c = 0;

void SysTick_Handler(void){
    if(delay_c > 0)
        delay_c--;
}

void delay_ms(uint16_t delay_t){
```



```

        delay_c = delay_t;
        while(delay_c){}
    }

    int main(void)
    {
        SysTick_Config(SystemCoreClock / 1000);
        LCD5110_GPIO_Init();
        LCD5110_Init();

        while(1)
        {
            LCD5110_Set_Pos(0,0);
            LCD5110_Write_Data(symb1);
            delay_ms(1000);

            LCD5110_Clear();
            delay_ms(1000);

        }
    }
}

```

LCDInit.h

```

#ifndef LCDINIT_H
#define LCDINIT_H

typedef enum
{
    OFF = 0,
    ON = 1
} Switch;

//LCD functions for working with registers
void LCD5110_CS(Switch sw);

//Reset
void LCD5110_RST(Switch sw);

//Data/Command select
void LCD5110_DC(Switch sw);

//Master Out Slave In
void LCD5110_MOSI(Switch sw);

//Serial Clock
void LCD5110_SCK(Switch sw);

void LCD5110_Init(void);

#endif

```

LCDInit.c

```

#include "LCDInit.h"
#include "LCD5110.h"
#include <stm32f407xx.h>

void LCD5110_CS(Switch sw)

```

```

{
    if (sw)
        GPIOC->ODR |= GPIO_ODR_ODR_9;
    else
        GPIOC->ODR &= ~GPIO_ODR_ODR_9;
}
//функція яка працює з піном RST, який відповідає за перезавантаження дисплею
void LCD5110_RST(Switch sw)
{
    if (sw)
        GPIOC->ODR |= GPIO_ODR_ODR_8;
    else
        GPIOC->ODR &= ~GPIO_ODR_ODR_8;
}

//функція яка працює з піном DC, який відповідає за режим вводу - виводу даних або
//вводу команд (LOW - дані, HIGH - команди)
void LCD5110_DC(Switch sw)
{
    if (sw)
        GPIOC->ODR |= GPIO_ODR_ODR_7;
    else
        GPIOC->ODR &= ~GPIO_ODR_ODR_7;
}

//Функція для передачі даних від веденого пристрою до ведучого
// Master In Slave Out
void LCD5110_MOSI(Switch sw)
{
    if (sw)
        GPIOC->ODR |= GPIO_ODR_ODR_6;
    else
        GPIOC->ODR &= ~GPIO_ODR_ODR_6;
}

//функція для передачі послідовного тактового сигналу для веденого пристрою
void LCD5110_SCK(Switch sw)
{
    if (sw)
        GPIOA->ODR |= GPIO_ODR_ODR_8;
    else
        GPIOA->ODR &= ~GPIO_ODR_ODR_8;
}

//ініціалізація регістрів для роботи із дисплеєм
void LCD5110_Init(void)
{
    LCD5110_DC(ON);
    LCD5110_MOSI(ON);
    LCD5110_SCK(ON);
    LCD5110_CS(ON);

    LCD5110_RST(OFF);
    LCD5110_RST(ON);
}

```

InitGPIO.h

```
#ifndef INITGPIO_H
```

```
#define INITGPIO_H

void LCD5110_GPIO_Init(void);

#endif
```

InitGPIO.c

```
#include <stm32f407xx.h>
#include "InitGPIO.h"

//Сконфігурував порти GPIOA та порт GPIOC – для роботи зі дисплеєм.

void LCD5110_GPIO_Init(void)
{
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN;

    GPIOA->MODER |= GPIO_MODER_MODER8_0;
    GPIOA->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR8;
    GPIOA->PUPDR = 0;
    GPIOA->OTYPER = 0;

    GPIOC->MODER |= GPIO_MODER_MODER7_0 | GPIO_MODER_MODER7_0 |
GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0;
    GPIOC->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR8;
    GPIOC->PUPDR = 0;
    GPIOC->OTYPER = 0;
}
```

LCD5110.h

```
#ifndef LCD5110_H
#define LCD5110_H
#include <stdint.h>
#define H 64
#define W 12

void LCD5110_Write_Byte(uint8_t dat, uint8_t mode);
void LCD5110_Clear(void);
void LCD5110_Set_Pos(uint8_t X, uint8_t Y);
void LCD5110_Set_XY(uint8_t X, uint8_t Y);
void LCD5110_Write_Data(uint8_t map[H][W]);
#endif
```

LCD5110.c

```
#include <stm32f407xx.h>
#include "LCD5110.h"
#include "LCDInit.h"

//функція для встановлення позиції
void LCD5110_Set_Pos(uint8_t X, uint8_t Y)
{
    LCD5110_Write_Byte(0x40 | Y, 0);
    LCD5110_Write_Byte(0x80 | X, 0);
}

void LCD5110_Set_XY(uint8_t X, uint8_t Y)
```

```

{
    LCD5110_Write_Byte(0x40 | Y, 0);
    LCD5110_Write_Byte(0x80 | (X*6), 0);
}

//функція для запису байта із масиву зображення
void LCD5110_Write_Byte(uint8_t dat, uint8_t mode)
{
    LCD5110_CS(OFF);

    if (!mode)
        LCD5110_DC(OFF);
    else
        LCD5110_DC(ON);

    for (uint8_t i = 0; i < 8; ++i)
    {
        LCD5110_MOSI(dat & 0x80);
        dat <<= 1;
        LCD5110_SCK(OFF);
        LCD5110_SCK(ON);
    }

    LCD5110_CS(ON);
}

//Функція для очищення дисплею
void LCD5110_Clear(void)
{
    for (uint8_t i = 0; i < H; ++i)
        for (uint8_t j = 0; j < W; ++j)
            LCD5110_Write_Byte(0, 1);
}

//функція для виводу зображення на екран
void LCD5110_Write_Data(uint8_t map[H][W])
{
    for (uint8_t i = 0; i < H; ++i)
        for (uint8_t j = 0; j < W; ++j)
            LCD5110_Write_Byte(map[i][j], 1);
}

```

РЕЗУЛЬТАТИ



Рис. 3. Запис в регістр MOSI значення елемента масиву

ВИСНОВКИ

Виконуючи цю лабораторну роботу, я засвоїв принципи роботи інтерфейсу SPI та здобув практичні навички організації роботи мікроконтролера з периферійними пристроями через SPI за допомогою бібліотеки CMSIS. Ознайомився із принципом взаємодії з акселерометром LIS302DL та з дисплеєм ET-NOKIA LCD 5110. Написав програму для ініціалізації роботи інтерфейсу SPI та зчитування значень акселерометра та написав функції для взаємодії з дисплеєм. Реалізував згідно індивідуального завдання функцію для очищення дисплею.