Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

Про виконання лабораторної роботи № 2
«Робота з флеш пам'яттю на прикладі SST25VF016B»
з дисципліни: «Програмування Мікроконтролерів»

Лектор:
доцент каф. ПЗ
Марусенкова Т.А.

Виконав:
ст. групи ПЗ-42
Бурець В.В.

Прийняв:
доцент каф. ПЗ
Марусенкова Т.А.

«___»_____2022р.
∑ = ___ _____

Львів – 2022

Мета роботи: Навчитися організовувати взаємодію між мікроконтролером та флеш пам’ятю SST25VF016B за інтерфейсом SPI, а також закріпити навички роботи з технічною документацією.

ЗАВДАННЯ

Адаптувати наведені у методичних вказівках надані коди для відповідної однієї з перелічених бібліотек: SPL, HAL або CMSIS Driver. Використати бібліотеку згідно з індивідуальним завданням.

Варіант 1: AAI – Yes; SPL; N=10

ХІД РОБОТИ

Налаштування інтерфейсу SPI.

Init.h

```
#ifndef inith
	#define inith

		#include "main.h"

		void h_drv_SPI_Initialization (void);
		/*Керування лінією CS (CE) для деактивації пам'яті */
		void h_drv_SPI_CS_Disable (void);

		/*Керування лінією CS (CE) для активації пам'яті */
		void h_drv_SPI_CS_Enable (void);

	#endif

	#include "init.h"

	//ініціалізація SPI1
	void h_drv_SPI_Initialization (void)
	{
		GPIO_InitTypeDef	GPIO_Init_LED;
		SPI_InitTypeDef			SPI_Init_user;

		RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
		RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
		RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

		GPIO_Init_LED.GPIO_Pin = GPIO_Pin_5;
		GPIO_Init_LED.GPIO_Mode = GPIO_Mode_AF;
```

```
	GPIO_Init_LED.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init_LED.GPIO_OType = GPIO_OType_PP;
	GPIO_Init_LED.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOA, &GPIO_Init_LED);

	GPIO_Init_LED.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_5;
	GPIO_Init_LED.GPIO_Mode = GPIO_Mode_AF;
	GPIO_Init_LED.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init_LED.GPIO_OType = GPIO_OType_PP;
	GPIO_Init_LED.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOB, &GPIO_Init_LED);

	GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource4, GPIO_AF_SPI1);
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource5, GPIO_AF_SPI1);


	GPIO_Init_LED.GPIO_Pin = GPIO_Pin_7;
	GPIO_Init_LED.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_Init_LED.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init_LED.GPIO_OType = GPIO_OType_PP;
	GPIO_Init_LED.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOD, &GPIO_Init_LED);

	h_drv_SPI_CS_Disable();

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
	SPI_Init_user.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	SPI_Init_user.SPI_Mode = SPI_Mode_Master;
	SPI_Init_user.SPI_DataSize = SPI_DataSize_8b;
	SPI_Init_user.SPI_CPOL = SPI_CPOL_High;
	SPI_Init_user.SPI_CPHA = SPI_CPHA_2Edge;
	SPI_Init_user.SPI_NSS = SPI_NSS_Soft;
	SPI_Init_user.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
	SPI_Init_user.SPI_FirstBit = SPI_FirstBit_MSB;
	SPI_Init_user.SPI_CRCPolynomial = 7;
	SPI_Init(SPI1, &SPI_Init_user);
	SPI_Cmd(SPI1, ENABLE);
}

/*Керування лінією CS (CE) для деактивації пам'яті */
//sFLASH_CS_HIGH()
void h_drv_SPI_CS_Disable ()
{
	GPIO_SetBits(GPIOD, GPIO_Pin_7);
}
```

```
/*Керування лінією CS (CE) для активації пам'яті */
//sFLASH_CS_LOW();
void h_drv_SPI_CS_Enable (void)
{
	GPIO_ResetBits(GPIOD, GPIO_Pin_7);
}


//ініціалізація SPI2

#include "init.h"
void h_drv_SPI_InitializationSPI2 (void)
{
	GPIO_InitTypeDef	GPIO_Init_LED;
	SPI_InitTypeDef			SPI_Init_user;

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

	GPIO_Init_LED.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
	GPIO_Init_LED.GPIO_Mode = GPIO_Mode_AF;
	GPIO_Init_LED.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init_LED.GPIO_OType = GPIO_OType_PP;
	GPIO_Init_LED.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOB, &GPIO_Init_LED);

	GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_SPI2);
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_SPI2);
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_SPI2);


	GPIO_Init_LED.GPIO_Pin = GPIO_Pin_12;
	GPIO_Init_LED.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_Init_LED.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init_LED.GPIO_OType = GPIO_OType_PP;
	GPIO_Init_LED.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOD, &GPIO_Init_LED);

	h_drv_SPI_CS_Disable();

	RCC_APB2PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
	SPI_Init_user.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	SPI_Init_user.SPI_Mode = SPI_Mode_Master;
	SPI_Init_user.SPI_DataSize = SPI_DataSize_8b;
```

```
	SPI_Init_user.SPI_CPOL = SPI_CPOL_High;
	SPI_Init_user.SPI_CPHA = SPI_CPHA_2Edge;
	SPI_Init_user.SPI_NSS = SPI_NSS_Soft;
	SPI_Init_user.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
	SPI_Init_user.SPI_FirstBit = SPI_FirstBit_MSB;
	SPI_Init_user.SPI_CRCPolynomial = 7;
	SPI_Init(SPI2, &SPI_Init_user);
	SPI_Cmd(SPI2, ENABLE);
}

/*Керування лінією CS (CE) для деактивації пам'яті */
//sFLASH_CS_HIGH()
void h_drv_SPI_CS_Disable ()
{
	GPIO_SetBits(GPIOD, GPIO_Pin_12);
}

/*Керування лінією CS (CE) для активації пам'яті */
//sFLASH_CS_LOW();
void h_drv_SPI_CS_Enable (void)
{
	GPIO_ResetBits(GPIOD, GPIO_Pin_12);
}


#ifndef flash_h
#define flash_h

	#include "stm32f4xx.h"
	#include "stm32f4xx_gpio.h"
	#include "stm32f4xx_rcc.h"
	#include "stm32f4xx_exti.h"
	#include "stm32f4xx_syscfg.h"
	#include "misc.h"
	#include "stm32f4xx.h"

	#define sFLASH_WIP_FLAG 100

	#define sFLASH_CMD_WREN 0x06
	#define sFLASH_CMD_WRITE 0x10

	#define sFLASH_SPI_PAGESIZE 4000
	#define sFLASH_SPI SPI2
	#define TIME_OUT 10000

	uint8_t sFLASH_SendByte(uint8_t byte);
```

```
	extern volatile uint16_t timeFromStart;

#endif


#include "flash_memory.h"
uint8_t sFLASH_SendByte(uint8_t byte)
{
	int time = timeFromStart;
 /*!< Loop while DR register in not empty */
 while (SPI_I2S_GetFlagStatus(sFLASH_SPI, SPI_I2S_FLAG_TXE) == RESET){
		if(timeFromStart - time > TIME_OUT){
			return 0x0;
		}
	};

 /*!< Send byte through the SPI2 peripheral */
 SPI_I2S_SendData(sFLASH_SPI, byte);
	time = timeFromStart;
 /*!< Wait to receive a byte */
 while (SPI_I2S_GetFlagStatus(sFLASH_SPI, SPI_I2S_FLAG_RXNE) == RESET){
		if(timeFromStart - time > TIME_OUT){
			return 0x0;
		}
	};

 return SPI_I2S_ReceiveData(sFLASH_SPI);
}




#ifndef flash_read_h
#define flash_read_h

 #include "main.h"
 #include "init.h"
 #include "flash_memory.h"

 #define sFLASH_DUMMY_BYTE 0x00
 #define sFLASH_CMD_READ 0x0B

 int sFLASH_ReadBuffer(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t NumByteToRead);
 void sFLASH_StartReadSequence(uint32_t ReadAddr);
 uint8_t sFLASH_ReadByte(void);
```

```
#endif

#include "flash_read.h"


int sFLASH_ReadBuffer(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t NumByteToRead)
{
 /*!< Select the FLASH: Chip Select low */
 h_drv_SPI_CS_Enable();

 /*!< Send "Read from Memory " instruction */
 int status = sFLASH_SendByte(sFLASH_CMD_READ);
        if(status == 0) return 0;
 /*!< Send ReadAddr high nibble address byte to read from */
 status = sFLASH_SendByte((ReadAddr & 0xFF0000) >> 16);
        if(status == 0) return 0;
 /*!< Send ReadAddr medium nibble address byte to read from */
 status = sFLASH_SendByte((ReadAddr& 0xFF00) >> 8);
        if(status == 0) return 0;
 /*!< Send ReadAddr low nibble address byte to read from */
 status = sFLASH_SendByte(ReadAddr & 0xFF);
        if(status == 0) return 0;

 while (NumByteToRead--) /*!< while there is data to be read */
 {
  /*!< Read a byte from the FLASH */
  *pBuffer = sFLASH_SendByte(sFLASH_DUMMY_BYTE);
  /*!< Point to the next location where the byte read will be saved */
  pBuffer++;
 }

 /*!< Deselect the FLASH: Chip Select high */
 h_drv_SPI_CS_Disable();
        return 1;
}


void sFLASH_StartReadSequence(uint32_t ReadAddr)
{
 /*!< Select the FLASH: Chip Select low */
 h_drv_SPI_CS_Enable();

 /*!< Send "Read from Memory " instruction */
 sFLASH_SendByte(sFLASH_CMD_READ);
```

```
 /*!< Send the 24-bit address of the address to read from ----------*/
 /*!< Send ReadAddr high nibble address byte */
 sFLASH_SendByte((ReadAddr & 0xFF0000) >> 16);
 /*!< Send ReadAddr medium nibble address byte */
 sFLASH_SendByte((ReadAddr& 0xFF00) >> 8);
 /*!< Send ReadAddr low nibble address byte */
 sFLASH_SendByte(ReadAddr & 0xFF);
}

uint8_t sFLASH_ReadByte(void)
{
 return (sFLASH_SendByte(sFLASH_DUMMY_BYTE));
}

#ifndef flash_write_h
#define flash_write_h

        #include "flash_memory.h"
        #include "init.h"

        #define sFLASH_CMD_RDSR 0x05
        #define sFLASH_DUMMY_BYTE 0x00

        int sFLASH_WriteBuffer(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite);
        int sFLASH_WritePage(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite);
        int sFLASH_WaitForWriteEnd(void);
        void sFLASH_WriteEnable(void);

#endif

#include "flash_write.h"

int sFLASH_WriteBuffer(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)
{
          /*!< Enable the write access to the FLASH */
 sFLASH_WriteEnable();

 /*!< Select the FLASH: Chip Select low */
 h_drv_SPI_CS_Enable();

        h_drv_SPI_Write_Byte(0xAD);

        h_drv_SPI_CS_Disable();

 int status = 0;
```

```
	while (NumByteToWrity--){
		status = sFlash_Write_Byte(pBuffer, WriteAddr);

		if(status == 0){
			return 0;
		}

		pBuffer++;
		WriteAddr++;
	}

	return 1;
}

int sFLASH_Write_Byte(uint8_t* pBuffer, uint32_t WriteAddr)
{
 /*!< Enable the write access to the FLASH */
 sFLASH_WriteEnable();

 /*!< Select the FLASH: Chip Select low */
 h_drv_SPI_CS_Enable();

	status = sFLASH_SendByte(*pBuffer);
 if(status == 0) return 0;

 /*!< Wait the end of Flash writing */
 status = sFLASH_WaitForWriteEnd();
	if(status == 0){
		return 0;
	}

	/*!< Deselect the FLASH: Chip Select high */
 h_drv_SPI_CS_Disable();

	return 1;
}

int sFLASH_WaitForWriteEnd(void)
{
 uint8_t flashstatus = 0;
 h_drv_SPI_CS_Enable();

 /*!< Send "Read Status Register" instruction */
 int res = sFLASH_SendByte(sFLASH_CMD_RDSR);
	if(res == 0) return 0;
	int time = timeFromStart;
```

```
 /*!< Loop as long as the memory is busy with a write cycle */
 do
 {
  /*!< Send a dummy byte to generate the clock needed by the FLASH
  and put the value of the status register in FLASH_Status variable */
  flashstatus = sFLASH_SendByte(sFLASH_DUMMY_BYTE);
                    if(timeFromStart - time > TIME_OUT){
                              return 0;
                    }
 }
 while ((flashstatus & sFLASH_WIP_FLAG) == SET); /* Write in progress */

 /*!< Deselect the FLASH: Chip Select high */
 h_drv_SPI_CS_Disable();
          return 1;
}

void sFLASH_WriteEnable(void)
{
  h_drv_SPI_CS_Enable();

 /*!< Send "Write Enable" instruction */
 sFLASH_SendByte(sFLASH_CMD_WREN);

 h_drv_SPI_CS_Disable();
}
```

Функція main.

```
#include "main.h"

int main(){
          SystemInit();
          SysTick_Config(SystemCoreClock/1000);
          h_drv_SPI_Initialization();

          int addr = 0x028A; // 10 * 64 + 10 = 28A 10блок + 10 байт
          uint8_t data[SIZE] = {0};

          int status = sFLASH_ReadBuffer(data, addr, SIZE);

          if(status == 0){
                    return -1;
          }
```

```
        status = sFLASH_WriteBuffer(data, addr, SIZE);

        if(status == 0){
                return 1;
        }
}

volatile uint16_t timeFromStart = 0;
void SysTick_Handler(void){
        timeFromStart++;
}
```

ВИСНОВОК

Під час виконання лабораторної роботи, перш за все, я освоїв принципи роботи інтерфейсу SPI. Дізнався як організувати взаємодію мікроконтролера з периферійними пристроями через SPI за допомогою бібліотеки SPL. Організував взаємодію з флеш пам’яттю SST25VF016B, використавши бібліотеку SPL. З документації дізнався про налаштування Auto Address Increment.