Лабораторна робота №4

Робота з кастомними бібліотеками. Виведення інформації на LED дисплей та I2C передача інформації

Мета: створення персональних бібліотек в STM32CubeIDE, вивід інформації на LED дисплей за допомогою I2C протоколу передачі інформації.

Теоретичні відомості

Комунікаційний протокол I2C вперше був створений компанією Philips Semiconductor у 1982 році, а з 2006 року розробникам та інженерам вбудованих технологій дозволено впроваджувати цей протокол без сплати будь-яких ліцензійних зборів.

I2C — це двопровідний протокол зв'язку, який зазвичай використовується для підключення низькошвидкісних пристроїв, таких як мікроконтролери, інтерфейси введення/виведення, аналого-цифрові та цифро-аналогові перетворювачі, EEPROM та інші периферійні пристрої у вбудованих системах. Один із цих проводів, відомий як SCL (Serial Clock), передає тактовий сигнал, а інший провід, відомий як SDA (Serial Data), дозволяє головним і підлеглим пристроям на шині надсилати й отримувати дані. Протокол I2C дозволяє підключити кілька підлеглих пристроїв до одного головного пристрою або кілька головних пристроїв, які керують одним або кількома підлеглими пристроями.

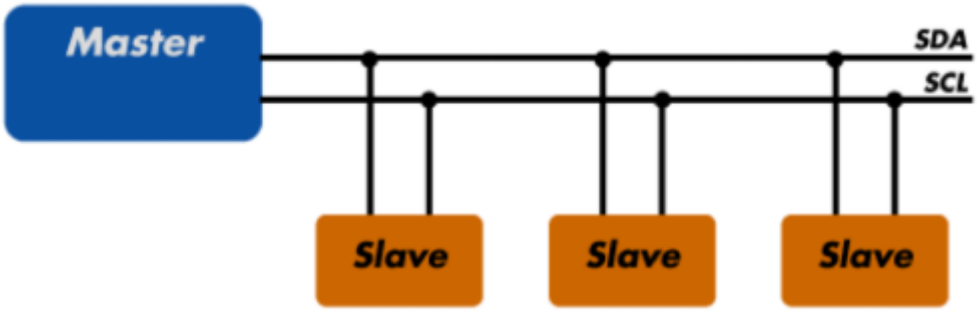

Рис.4.1 – I2C протокол.

Загалом, протокол I2C найкраще застосовувати в проектах, де простота конструкції та низькі витрати на виробництво важливіші за швидкість. Стандартна швидкість передачі даних, пов'язана з протоколом I2C, становить лише 100 Кбіт/с, хоча швидкість до 5 Мбіт/с можлива з певними типами пристроїв, налаштованих на використання I2C у «швидкому режимі» або «надшвидкому режимі». Схожий до I2C є SPI.

SPI — це чотирипровідний протокол зв'язку, який можна використовувати для підключення мікроконтролерів до різноманітних пристроїв на одній послідовній шині, включаючи датчики температури та тиску, аналого-цифрові та цифро-аналогові перетворювачі, пристрої пам'яті, РК-дисплеї тощо. Чотирипровідна конфігурація являє собою дуже логічний підхід до полегшення передачі даних по послідовній шині. Кожен з чотирьох проводів відповідає певному логічному сигналу:

1. SCLK (Serial Clock): Провід послідовного годинника передає сигнал синхронізації від головного пристрою до інших пристроїв на послідовній шині.
2. MOSI (головний вихід, підлеглий вхід): провід MOSI передає вихідні дані від головного пристрою до підлеглих пристроїв на послідовній шині.
3. MISO (головний вхід, підлеглий вихід): провід MISO передає вихідні дані від вибраного підлеглого пристрою до головного пристрою або мікроконтролера на послідовній шині.

4. SS (вибір підлеглого): на шині SPI має бути один головний пристрій, але може бути кілька підлеглих пристроїв. Головний пристрій може обмінюватися даними з усіма підлеглими пристроями, але підлеглі пристрої можуть надсилати дані лише головному, а не один одному. Головний пристрій використовує провід підлеглого вибору, щоб вибрати, з яким підлеглим пристроєм на шині він буде спілкуватися перед відправленням передачі даних.

Протокол SPI був розроблений для забезпечення високошвидкісної ініціалізації периферійних пристроїв пристрою на тій самій інтегральній схемі, що й мікроконтролер. Загалом, протокол SPI найкраще застосовувати в проектах, де швидкість є важливішою, ніж мінімізація витрат на виробництво, і коли немає обмежень у використанні додаткових проводів, необхідних для полегшення зв'язку SPI. Ви також можете дослідити відмінності між UART і SPI для вбудованих систем .

I2C вигідний своєю простотою та легкістю додавання додаткових пристроїв до шини. Оскільки для зв'язку I2C потрібні лише два дроти, цей протокол добре підходить для плат із багатьма пристроями. У свою чергу, це допомагає зменшити вартість і складність схеми. Крім того, I2C пропонує контроль потоку та обробку помилок.
Однак I2C використовує підпорядковану адресацію та підтвердження у своїй схемі зв'язку, що може бути недоліком у певних випадках, коли це додає додаткової складності та накладних витрат.

З іншого боку, SPI перевершує швидкість. Пристрої SPI використовують двотактні драйвери, які забезпечують чудову швидкість і цілісність сигналу порівняно з відкритими лініями стоку, які використовуються в протоколі I2C. Крім того, SPI підтримує повнодуплексний зв'язок, коли і головний, і підлеглий можуть надсилати дані одночасно через лінії MOSI та MISO.

Однак, на відміну від I2C, додавання додаткових пристроїв до шини може збільшити складність плати. Це пояснюється тим, що кожному підпорядкованому пристрої потрібна власна лінія вибору підлеглого пристрою, тому кількість проводів, необхідних для зв'язку, збільшується з кожним пристроєм.

Хід виконання роботи

1. Створити свою бібліотеку та налаштувати середовище:

1.1 В середині папки **Core/Inc** створити файл liquidcrystal_i2c.h.

1.2 В середині папки **Core/Src** створити файл liquidcrystal_i2c.c.

1.3 В вкладці Connectivity увімкнути I2C рис.4.2 та зберегти налаштування.

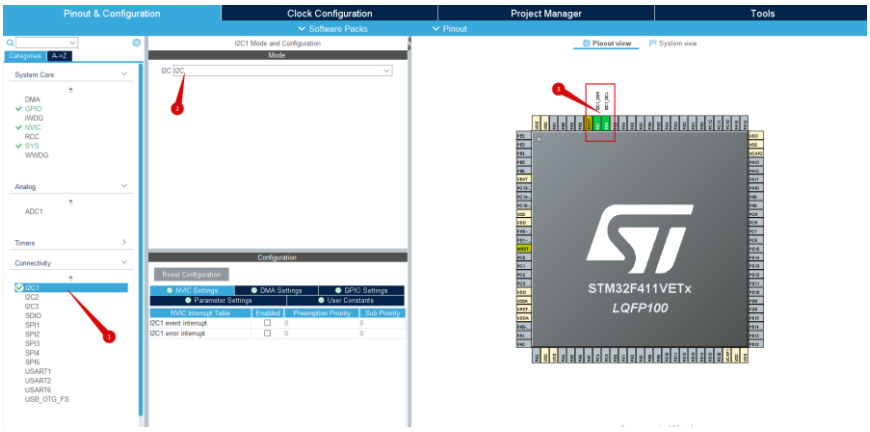

Рис.4.2 – Підключення I2C

1.4 Під’єднати до плати LED дисплей використовуючи I2C протокол.

1.5 Завантажити створену бібліотеку до коду.

1.5 Перевірти роботу бібліотеки та правильність підключення за допомогою коду:

```
/* USER CODE BEGIN 2 */
	HD44780_Init(2);
	HD44780_Clear();
	HD44780_SetCursor(0,0);
	HD44780_PrintStr("HELLO");
	HD44780_SetCursor(10,1);
	HD44780_PrintStr("WORLD");
	HAL_Delay(2000);
 /* USER CODE END 2 */
```

При правильному виконані усіх дій на LED дисплеї буде виведено HELLO на початку першого рядка та WORLD на 10ій позиції 2го рядка.

1.6 Протестувати наступні функції:

- **HD44780_Init(uint8_t rows)** – Ініціалізує дисплей у 4-бітному режимі, встановлює параметри відображення та створює спеціальні символи.
- **HD44780_Clear()** – Очищує екран дисплея.
- **HD44780_Home()** – Переміщує курсор у початкову позицію (0,0).
- **HD44780_NoDisplay()** – Вимикає відображення символів на дисплеї.
- **HD44780_Display()** – Увімкнення відображення символів.
- **HD44780_NoBlink()** – Вимикає миготіння курсора.
- **HD44780_Blink()** – Увімкнення миготіння курсора.
- **HD44780_NoCursor()** – Вимикає відображення курсора.
- **HD44780_Cursor()** – Увімкнення курсора.
- **HD44780_ScrollDisplayLeft()** – Прокручує весь дисплей вліво.
- **HD44780_ScrollDisplayRight()** – Прокручує весь дисплей вправо.
- **HD44780_LeftToRight()** – Встановлює напрямок виводу тексту зліва направо.
- **HD44780_RightToLeft()** – Встановлює напрямок виводу тексту справа наліво.
- **HD44780_NoBacklight()** – Вимикає підсвічування дисплея.
- **HD44780_Backlight()** – Увімкнення підсвічування дисплея.
- **HD44780_AutoScroll()** – Вмикає автоматичну прокрутку тексту.
- **HD44780_NoAutoScroll()** – Вимикає автоматичну прокрутку тексту.
- **HD44780_CreateSpecialChar(uint8_t, uint8_t[])** – Створює спеціальний символ у CGRAM дисплея.
- **HD44780_PrintSpecialChar(uint8_t)** – Виводить спеціальний символ на екран.
- **HD44780_SetCursor(uint8_t, uint8_t)** – Встановлює курсор у вказану позицію.
- **HD44780_SetBacklight(uint8_t new_val)** – Увімкнення або вимикання підсвічування дисплея.
- **HD44780_LoadCustomCharacter(uint8_t char_num, uint8_t *rows)** – Завантажує спеціальний символ у CGRAM.
- **HD44780_PrintStr(const char[])** – Виводить рядок тексту на дисплей.

Індивідуальні завдання

1. Реалізувати декілька режимів роботи LED дисплея(при натисканні користувацької кнопки відбувається зміна режиму):
   1.1 Статичне виведення власного імені та групи на дисплеї(номер варіанта це початкова позиція виведення інформації).
   1.2 Статичне виведення з вимкненим підсвічуванням дисплею.
   1.3 Статичне виведення з увімкненим курсором
   1.4 Динамічний скрол вліво для непарних варіантів та вправо для парних варіантів.

Контрольні запитання

1. Що таке I2C і для чого він використовується?
2. Які сигнали використовує I2C і їх призначення?
3. Як формується стартовий (Start) і стоповий (Stop) сигнали в I2C?
4. Які основні відмінності I2C від SPI?
5. Як підключати кілька підлеглих пристроїв із однаковою адресою на одну I2C-шину?
6. Як перевірити, чи підлеглий пристрій відповідає на запит?

```
#include "liquidcrystal_i2c.h"

extern I2C_HandleTypeDef hi2c1;

uint8_t dpFunction;
uint8_t dpControl;
uint8_t dpMode;
uint8_t dpRows;
uint8_t dpBacklight;

static void SendCommand(uint8_t);
static void SendChar(uint8_t);
static void Send(uint8_t, uint8_t);
static void Write4Bits(uint8_t);
static void ExpanderWrite(uint8_t);
static void PulseEnable(uint8_t);
static void DelayInit(void);
static void DelayUS(uint32_t);

uint8_t special1[8] = {
	0b00000,
	0b11001,
	0b11011,
	0b00110,
	0b01100,
	0b11011,
	0b10011,
	0b00000
};

uint8_t special2[8] = {
	0b11000,
	0b11000,
	0b00110,
	0b01001,
	0b01000,
	0b01001,
	0b00110,
	0b00000
};

void HD44780_Init(uint8_t rows)
{
  dpRows = rows;

  dpBacklight = LCD_BACKLIGHT;

  dpFunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;

  if (dpRows > 1)
  {
    dpFunction |= LCD_2LINE;
  }
```

```
  else
  {
    dpFunction |= LCD_5x10DOTS;
  }

  /* Wait for initialization */
  DelayInit();
  HAL_Delay(50);

  ExpanderWrite(dpBacklight);
  HAL_Delay(1000);

  /* 4bit Mode */
  Write4Bits(0x03 << 4);
  DelayUS(4500);

  Write4Bits(0x03 << 4);
  DelayUS(4500);

  Write4Bits(0x03 << 4);
  DelayUS(4500);

  Write4Bits(0x02 << 4);
  DelayUS(100);

  /* Display Control */
  SendCommand(LCD_FUNCTIONSET | dpFunction);

  dpControl = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
  HD44780_Display();
  HD44780_Clear();

  /* Display Mode */
  dpMode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
  SendCommand(LCD_ENTRYMODESET | dpMode);
  DelayUS(4500);

  HD44780_CreateSpecialChar(0, special1);
  HD44780_CreateSpecialChar(1, special2);

  HD44780_Home();
}

void HD44780_Clear()
{
  SendCommand(LCD_CLEARDISPLAY);
  DelayUS(2000);
}

void HD44780_Home()
{
  SendCommand(LCD_RETURNHOME);
  DelayUS(2000);
}
```

```
void HD44780_SetCursor(uint8_t col, uint8_t row)
{
  int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
  if (row >= dpRows)
  {
    row = dpRows-1;
  }
  SendCommand(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}

void HD44780_NoDisplay()
{
  dpControl &= ~LCD_DISPLAYON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Display()
{
  dpControl |= LCD_DISPLAYON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_NoCursor()
{
  dpControl &= ~LCD_CURSORON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Cursor()
{
  dpControl |= LCD_CURSORON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_NoBlink()
{
  dpControl &= ~LCD_BLINKON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Blink()
{
  dpControl |= LCD_BLINKON;
  SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_ScrollDisplayLeft(void)
{
  SendCommand(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}

void HD44780_ScrollDisplayRight(void)
{
```

```
  SendCommand(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}

void HD44780_LeftToRight(void)
{
  dpMode |= LCD_ENTRYLEFT;
  SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_RightToLeft(void)
{
  dpMode &= ~LCD_ENTRYLEFT;
  SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_AutoScroll(void)
{
  dpMode |= LCD_ENTRYSHIFTINCREMENT;
  SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_NoAutoScroll(void)
{
  dpMode &= ~LCD_ENTRYSHIFTINCREMENT;
  SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_CreateSpecialChar(uint8_t location, uint8_t charmap[])
{
  location &= 0x7;
  SendCommand(LCD_SETCGRAMADDR | (location << 3));
  for (int i=0; i<8; i++)
  {
    SendChar(charmap[i]);
  }
}

void HD44780_PrintSpecialChar(uint8_t index)
{
  SendChar(index);
}

void HD44780_LoadCustomCharacter(uint8_t char_num, uint8_t *rows)
{
  HD44780_CreateSpecialChar(char_num, rows);
}

void HD44780_PrintStr(const char c[])
{
  while(*c) SendChar(*c++);
}

void HD44780_SetBacklight(uint8_t new_val)
{
```

```
  if(new_val) HD44780_Backlight();
  else HD44780_NoBacklight();
}

void HD44780_NoBacklight(void)
{
  dpBacklight=LCD_NOBACKLIGHT;
  ExpanderWrite(0);
}

void HD44780_Backlight(void)
{
  dpBacklight=LCD_BACKLIGHT;
  ExpanderWrite(0);
}

static void SendCommand(uint8_t cmd)
{
  Send(cmd, 0);
}

static void SendChar(uint8_t ch)
{
  Send(ch, RS);
}

static void Send(uint8_t value, uint8_t mode)
{
  uint8_t highnib = value & 0xF0;
  uint8_t lownib = (value<<4) & 0xF0;
  Write4Bits((highnib)|mode);
  Write4Bits((lownib)|mode);
}

static void Write4Bits(uint8_t value)
{
  ExpanderWrite(value);
  PulseEnable(value);
}

static void ExpanderWrite(uint8_t _data)
{
  uint8_t data = _data | dpBacklight;
  HAL_I2C_Master_Transmit(&hi2c1, DEVICE_ADDR, (uint8_t*)&data, 1, 10);
}

static void PulseEnable(uint8_t _data)
{
  ExpanderWrite(_data | ENABLE);
  DelayUS(20);

  ExpanderWrite(_data & ~ENABLE);
  DelayUS(20);
}
```

```
static void DelayInit(void)
{
  CoreDebug->DEMCR &= ~CoreDebug_DEMCR_TRCENA_Msk;
  CoreDebug->DEMCR |=  CoreDebug_DEMCR_TRCENA_Msk;

  DWT->CTRL &= ~DWT_CTRL_CYCCNTENA_Msk; //~0x00000001;
  DWT->CTRL |=  DWT_CTRL_CYCCNTENA_Msk; //0x00000001;

  DWT->CYCCNT = 0;

  /* 3 NO OPERATION instructions */
  __ASM volatile ("NOP");
  __ASM volatile ("NOP");
  __ASM volatile ("NOP");
}

static void DelayUS(uint32_t us) {
  uint32_t cycles = (SystemCoreClock/1000000L)*us;
  uint32_t start = DWT->CYCCNT;
  volatile uint32_t cnt;

  do
  {
    cnt = DWT->CYCCNT - start;
  } while(cnt < cycles);
}
```

Додаток 2 - **liquidcrystal_i2c.h**

```
#ifndef LIQUIDCRYSTAL_I2C_H_
#define LIQUIDCRYSTAL_I2C_H_

#include "stm32f4xx_hal.h"

/* Command */
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

/* Entry Mode */
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

/* Display On/Off */
#define LCD_DISPLAYON 0x04
```

```
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

/* Cursor Shift */
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

/* Function Set */
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

/* Backlight */
#define LCD_BACKLIGHT 0x08
#define LCD_NOBACKLIGHT 0x00

/* Enable Bit */
#define ENABLE 0x04

/* Read Write Bit */
#define RW 0x0

/* Register Select Bit */
#define RS 0x01

/* Device I2C Address */
#define DEVICE_ADDR     (0x27 << 1)

void HD44780_Init(uint8_t rows);
void HD44780_Clear();
void HD44780_Home();
void HD44780_NoDisplay();
void HD44780_Display();
void HD44780_NoBlink();
void HD44780_Blink();
void HD44780_NoCursor();
void HD44780_Cursor();
void HD44780_ScrollDisplayLeft();
void HD44780_ScrollDisplayRight();
void HD44780_PrintLeft();
void HD44780_PrintRight();
void HD44780_LeftToRight();
void HD44780_RightToLeft();
void HD44780_ShiftIncrement();
void HD44780_ShiftDecrement();
void HD44780_NoBacklight();
```

```
void HD44780_Backlight();
void HD44780_AutoScroll();
void HD44780_NoAutoScroll();
void HD44780_CreateSpecialChar(uint8_t, uint8_t[]);
void HD44780_PrintSpecialChar(uint8_t);
void HD44780_SetCursor(uint8_t, uint8_t);
void HD44780_SetBacklight(uint8_t new_val);
void HD44780_LoadCustomCharacter(uint8_t char_num, uint8_t *rows);
void HD44780_PrintStr(const char[]);

#endif /* LIQUIDCRYSTAL_I2C_H_ */
```