Міністерство освіти та науки України
Чорноморський національний університет імені Петра Могили

Я. М. Крайник

Комп'ютерні системи

Методичні вказівки до виконання лабораторних робіт

Випуск 258

Миколаїв – 2018

УДК 004.3(076)
К 78

Рекомендовано до друку вченою радою ЧНУ ім. Петра Могили (витяг з протоколу № 3 від 8 листопада 2018 року).

***Рецензенти*:**
Мусієнко М. П. д-р техн. наук, професор кафедри комп'ютерної інженерії Чорноморського національного університету ім. Петра Могили;
Пузирьов С. В. канд. фіз.-мат. наук, доцент кафедри комп'ютерної інженерії Чорноморського національного університету ім. Петра Могили.

К 78 **Крайник Я. М.** Комп'ютерні системи : метод. вказ. до виконання лабораторних робіт. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2018. 48 с. (Методична серія ; вип. 258).

Методичні вказівки містять завдання для виконання лабораторних робіт з дисципліни «Комп'ютерні системи». Наведено необхідний теоретичний матеріал, а також приклад програми, який можна використовувати для реалізації програми відповідно до завдання.

Методичні вказівки призначені для студентів денної форми навчання, які навчаються за освітньо-кваліфікаційним рівнем «бакалавр» за напрямом підготовки 6.050102 «Комп'ютерна інженерія».

УДК 004.3(076)

ISSN 1811–492X

ЗМІСТ

ВСТУП

Метою дисципліни «Комп'ютерні системи» є вивчення програмних та апаратних засобів для побудови комп'ютерних систем.

Робочою програмою дисципліни передбачено проведення лабораторних робіт. Завдання лабораторних робіт призначені для надання студентам практичних навичок щодо побудови апаратно-програмних комплексів на базі мікроконтролерів.

Архітектура ARM для сучасних мікроконтролерів є однією з найперспективніших та здатна запропонувати великі переваги навіть для простих пристроїв. Однією з платформ, які реалізують цю архітектуру, є пристрої STM32. Вони здатні забезпечувати як низьке енергоспоживання, так і достатню обчислювальну потужність для мікроконтролерних пристроїв. Відповідно, діапазон їх застосувань поширюється практично на весь сегмент пристроїв «Інтернету речей», де необхідно забезпечити взаємодію (контроль, зчитування, обробка даних) із зовнішніми пристроями. Саме тому дослідження можливостей подібних мікроконтролерів є актуальним та дозволяє розуміти тенденції, які мають місце в галузі вбудованих систем. Це стосується як апаратної складової, так і програмної.

У якості середовища для розробки використовується online-середовище mbed, яке надає не лише середовище, а і власну програмну інфраструктуру, організовану навколо окремої операційної системи mbed OS. Завдяки наявності спеціального завантажувача процес програмування значно спрощується та займає мінімальну кількість.

Лабораторні роботи мають комплексний характер та передбачають як роботу з апаратною складовою (збір схеми), так і програмною (безпосередньо програмування).

ЛАБОРАТОРНА РОБОТА № 1.
РОБОТА З ПОРТАМИ ВВЕДЕННЯ/ВИВЕДЕННЯ

Пройти реєстрацію на сайті https://developer.mbed.org.

Під час роботи в середовищі воно створює для вас проект та компілює його. У результаті отримуємо двійковий файл, який необхідно скачати та записати на плату. Більшість з плат, які пропонуються, при підключенні до комп'ютера відображаються як зовнішні носії, тому записати файл можна операціями копіювання та вставки.

Після реєстрації треба обрати пункт меню сайту, пов'язаний з вибором цільового апаратного забезпечення (Hardware/Boards).

Установити фільтр STMicroelectronics та обрати плату Nucleo-L053R8. Додати плату до списку своїх плат у середовищі (кнопка Add to my Compiler).

Найпростішою периферією, яка доступна для користувача, є порти введення/виведення. У цій роботі необхідно організувати керування станом портів виведення відповідно до стану порту введення (0 або 1).

У платформі mbed робота з портами реалізована за допомогою класів DigitalIn та DigitalOut. Вони прості та зрозумілі в керуванні. Для більш детального ознайомлення з можливостями таких класів рекомендується попередньо переглянути документацію (рис. 1.1), яку надає mbed, а також і приклад використання.

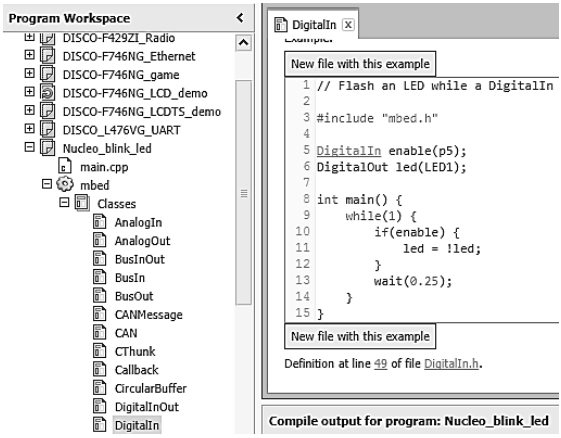


Рис. 1.1. Приклад використання класу в документації

Для того, щоб ініціалізувати екземпляри цих класів у них необхідно передати назву піну (ці назви визначені у .h-файлах проекту; зазвичай можна користуватись тими назвами, якими піни підписані безпосередньо на платі, проте у цьому необхідно попередньо переконатись на сторінці з інформацією про плату – рис. 1.2).

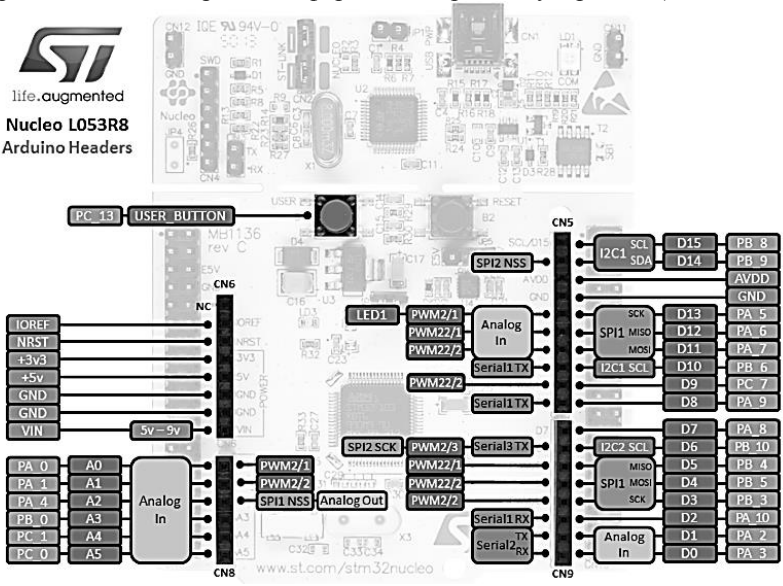


Рис. 1.2. Найменування пінів та деякі з їхніх доступних функцій

Для використання у програмі доступними є назви, що наведені в блакитних та зелених прямокутниках (білий шрифт).

Розглянемо базовий приклад, який запропоновано використати для виконання завдання.

Необхідно створити нову програму в середовищі (меню New…/ Program), параметри якої вказуються в діалоговому вікні (рис. 1.3). Також процес створення нової програми описаний у роботі з реалізацією ефектів на дисплеї.

Дуже зручно, що mbed дозволяє імпортувати доступні шаблони (пункт Template) для нової програми, оскільки в такому випадку будуть імпортовані також необхідні бібліотеки (для більш складних прикладів) та готовий певний код у файлі main.c.

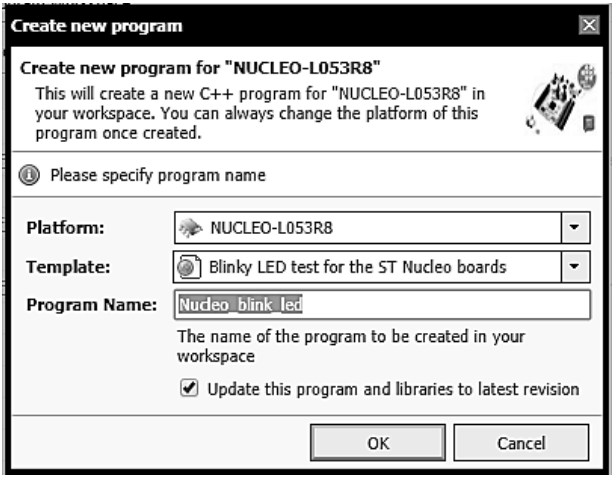


Рис. 1.3. Діалогове вікно налаштування нової програми

Додамо у створений файл main.c такий код:

```
#include "mbed.h"

DigitalOut led0(A0);
DigitalOut led1(D2);
DigitalOut led2(A2);
DigitalIn buttonIn(PA_1, PullNone);

int main() {
    while(1) {
        if (buttonIn == 1) {
            led0 = 1;
            led1 = 0;
            led2 = 1;
        } else if (buttonIn == 0) {
            led0 = 0;
            led1 = 1;
            led2 = 0;
        }
    }
}
```

Розглянемо організацію програми: у функції main розміщено нескінченний цикл while (1), тобто, програма виконуватиметься постійно, поки подається живлення на плату. Загалом можна сказати, що такий цикл буде присутній у всіх наступних програмах – змінюватиметься лише його внутрішня частина. Також наявна ініціалізація пінів: 3 – на виведення для світлодіодів, і 1 – для кнопки. У циклі перевіряється стан кнопки та відповідно до нього змінюється

стан виходів. Завдяки тому, що у класах перевантажено оператор приведення до int, об'єктами цих класів можна маніпулювати як цілими числами.

Завдання

Використовуючи функцію wait, що в якості параметра приймає число з плаваючою комою та призначена для організації затримки, реалізувати ефект по натисненні на кнопку, зібрати схему з 3 світлодіодами та 1 кнопкою за аналогією з наведеною (рис. 1.4). У випадку використання мезонінної плати (вставляється в роз'єм Arduino) рекомендується використовувати піни D9, D10, D11 для керування.

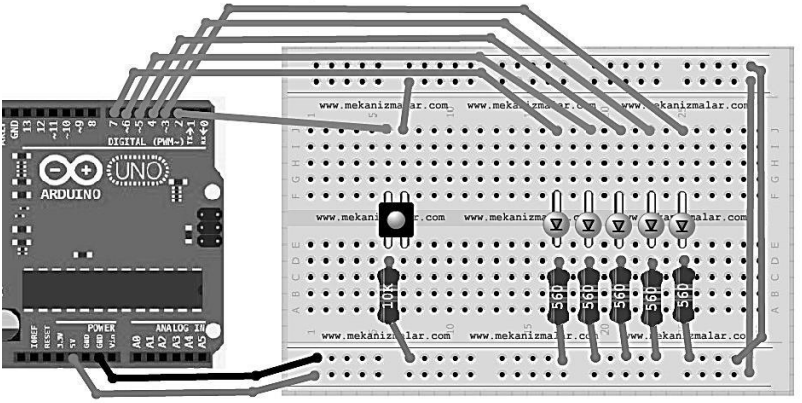


Рис. 1.4. Схема підключення компонентів

Увага! Перед підключенням живлення до зібраної схеми викладач повинен перевірити її коректність!

1. Реалізувати ефект, коли при натисненій кнопці вмикаються світлодіоди по одному, якщо натиснення немає – в іншому напрямку.
2. Реалізувати ефект, коли при натисненій кнопці вмикаються всі діоди, а потім вимикаються, якщо натиснення немає – спочатку все вимкнено, а потім вмикається по одному.
3. Якщо кнопка натиснена, то пристрій працює як світлофор, якщо відтиснута – почергово 2 блимання кожним світлодіодом.
4. Якщо кнопка натиснена, то почергово передається 3 світлодіодами сигнал SOS, якщо відтиснута – сигнал OSO.

ЛАБОРАТОРНА РОБОТА № 2.
ПОСЛІДОВНІ ІНТЕРФЕЙСИ.
ІНТЕРФЕЙС SPI. МІКРОСХЕМА MAX7219

У цій роботі необхідно ознайомитися з можливостями, які надає програмна реалізація інтерфейсу SPI у платформі mbed та реалізувати ефект на основі драйвера MAX7219, що може керувати світлодіодною матрицею 8x8. Рекомендована плата для виконання цієї роботи – STM32 Nucleo-L053R8, але може використовуватися будь-яка плата, у якій є SPI.

Serial Peripheral Interface (SPI) – послідовний синхронний інтерфейс для передачі даних. Є одним з найбільш поширених для керування зовнішніми пристроями, зняття інформації з датчиків тощо. Працює у режимі Master-Slave, тобто один пристрій виступає у ролі керувального – надсилає команди та дані, а інший їх приймає. У цьому випадку в ролі Master виступає плата STM, а в ролі підлеглого – MAX7219. Зазвичай, інтерфейс SPI передбачає використання 4 ліній – CLK (тактовий сигнал), MISO (Master Input Slave Output), MOSI (Master Output Slave Input) та CS або NSS (Chip Select – для вибору активного пристрою). У цьому випадку керувальний пристрій буде лише передавати дані, тому використовуються лише 3 лінії (MISO відсутня). Більш детальну інформацію про цей інтерфейс можна знайти за вказаним у дужках посиланням (https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus).

У mbed для використання цього послідовного інтерфейсу в режимі керувального пристрою є клас SPI. Загалом, порівняно з бібліотекою Hardware Abstract Library (HAL), він значно спрощує роботу з SPI. З іншого боку, якщо Вам необхідно досягнути максимальної швидкодії SPI, то доцільно використовувати HAL.

Живлення для модуля MAX7219 підключається безпосередньо з плати (виходи GND та 5V). Вихід MOSI підключіть до входу DIN на модулі. Назви інших пінів на платах для підключення збігаються. У прикладі використовуються піни, які належать до модуля SPI2. Мікросхема MAX7219 працює в режимі SPI-0 та має нестандартний алгоритм обробки сигналу CS, тому у прикладі під це виділено окремий пін.

Розглянемо приклад програми (рис. 2.1), яка пропонується як базова для виконання завдання:

```
#include "mbed.h"

#define REG_NO_OP          0x00 << 8
#define REG_DIGIT_0        0x01 << 8
#define REG_DIGIT_1        0x02 << 8
#define REG_DIGIT_2        0x03 << 8
#define REG_DIGIT_3        0x04 << 8
#define REG_DIGIT_4        0x05 << 8
#define REG_DIGIT_5        0x06 << 8
#define REG_DIGIT_6        0x07 << 8
#define REG_DIGIT_7        0x08 << 8
#define REG_DECODE_MODE    0x09 << 8
#define REG_INTENSITY      0x0A << 8
#define REG_SCAN_LIMIT     0x0B << 8
#define REG_SHUTDOWN       0x0C << 8
#define REG_DISPLAY_TEST   0x0F << 8

SPI spi_dev(PB_15, PB_14, PB_13);
DigitalOut cs(PC_4);

void sendData(uint16_t data);
void startCommunications(void);
void setDisplayData(uint8_t data[]);

int main() {
    uint8_t data[] = {0xff, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
    spi_dev.frequency(1000000);
    spi_dev.format(16, 0);
    cs = 1;
    startCommunications();
    setDisplayData(data);
    while(1) {

    }
}
void sendData(uint16_t data) {
    spi_dev.write(data);
    cs = 0;
    cs = 1;
}

void startCommunications(void) {
    sendData(REG_SHUTDOWN | 0x01);
    sendData(REG_DECODE_MODE | 0x00);
    sendData(REG_SCAN_LIMIT | 0x0F);
}
void setDisplayData(uint8_t data[]) {
    sendData(REG_DIGIT_0 | data[0]);
    sendData(REG_DIGIT_1 | data[1]);
    sendData(REG_DIGIT_2 | data[2]);
    sendData(REG_DIGIT_3 | data[3]);
    sendData(REG_DIGIT_4 | data[4]);
    sendData(REG_DIGIT_5 | data[5]);
    sendData(REG_DIGIT_6 | data[6]);
    sendData(REG_DIGIT_7 | data[7]);
}
```

Рис 2.1. Приклад програми

Керування модулем відбувається шляхом запису команд до регістрів. На початку файлу розміщені директиви #define, у яких вказується адреса даних регістрів. Найважливішими для відображення даних на матриці є 8 регістрів даних. У тому випадку, коли використовується режим без декодування (як у прикладі), якщо у регістрі певний біт має значення 1, то це означає, що відповідний цьому бітові світлодіод буде ввімкнений.

Увага! При відключенні модуля від живлення спостерігається ефект, коли усі 64 діоди ввімкнені (при повторному включенні). У такому разі рекомендується записати програму в мікроконтролер ще раз, після чого вимкнути та підключити живлення знову.

Завдання

1. Реалізувати виведення слова у вигляді анімації: з'являється перша літера, після чого вона зміщується та на її місці з'являється інша літера. Після закінчення слова дія повторюється.
2. Реалізувати поступове заповнення матриці по периметру і до центру – у результаті всі діоди будуть увімкнені.
3. Реалізувати обертання на матриці квадрата та трикутника. Спочатку оберт робить квадрат, потім – трикутник.
4. Реалізувати появу літери на матриці згори вниз, потім вона «провалюється» і все починається спочатку.

ЛАБОРАТОРНА РОБОТА № 3.
ШИРОТНО-ІМПУЛЬСНА МОДУЛЯЦІЯ (ШІМ).
ЗОВНІШНІ ПЕРЕРИВАННЯ

У даній роботі необхідно реалізувати ефект з використанням RGB-світлодіоду та кнопки, обробку натиснення якої треба організувати засобами mbed. Рекомендована плата для реалізації – STM32 Nucleo-L053R8.

Широтно-імпульсна модуляція (ШІМ, англ. Pulse Width Modulation – PWM) – один з найпоширеніших видів модуляції, який часто використовується для керування цифровими пристроями. Цей вид модуляції є цифровим (стани 1 та 0). ШІМ характеризується періодом та коефіцієнтом заповнення. Чим ближче значення коефіцієнта заповнення, тим більша напруга на виході. На такому принципі базується керування ШІМ для світлодіодів або двигунів, які не сприймають високу частоту ШІМ, проте керуються зміною напруги, яка є наслідком цього.

Платформа mbed дозволяє використовувати ШІМ через клас PwmOut. Конструктор цього класу приймає на вхід аргумент – назву піну, на якому необхідно організувати ШІМ. Проте, не кожен пін може використовуватись для генерації такого сигналу, тому треба попередньо переконатись, що є підтримка цієї функції за допомогою документації mbed (рис. 3.1).

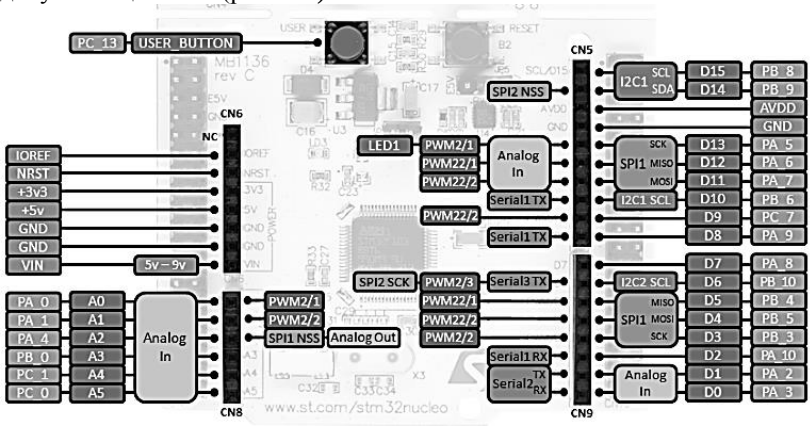


Рис. 3.1. Діаграма функціональних призначень пінів

Виводи, які доступні для організації ШІМ, позначені фіолетовим кольором (проте, як вже було сказано, не слід забувати, що у програмі

(рис. 3.2) доступні лише назви, наведені в зелених та блакитних прямокутниках). У цьому прикладі програми необхідно 3 виводи, тому використовуватимемо піни D6, D5, D4. Ці піни підключені до червоного, синього та зеленого входів (не забувайте про необхідність резисторів під час підключення, щоб обмежити струм).

Розглянемо приклад роботи з ШІМ (рис 3.2).

```
#include "mbed.h"

PwmOut pwm1(D6);
PwmOut pwm2(D5);
PwmOut pwm3(D4);

InterruptIn button(PC_13);
uint8_t dir;

void callback() {
    if (dir) dir = 0;
    else dir = 1;
}

int main() {
    pwm1 = 0.5; // red
    pwm2 = 0.6; // green
    pwm3 = 0.0; // blue
    button.rise(&callback);
    while(1) {
        wait(0.3);
        if (dir) {
            if (pwm3 < 0.9)
                pwm3 = pwm3 + 0.1;
        }
        else {
            if (pwm3 > 0.1)
                pwm3 = pwm3 - 0.1;
        }
    }
}
```

Рис 3.2. Приклад програми

Керування виходами ШІМ можна здійснювати як функціями pulsewidth та period та їхніми відповідниками для різних одиниць часу,

так і просто, присвоюючи числове значення від 0 до 1 змінній класу PwmOut. У другому випадку варто бути обережним, оскільки вихід за границі цього діапазону призведе до зависання програми. Також слід звернути увагу на екземпляр класу InterruptIn – button. Цей клас використовується для роботи із зовнішніми перериваннями. У цьому конкретному випадку задана реакція на фронт сигналу (зміна з 0 в 1) за допомогою функції rise, якій у якості параметра передається функція, яка буде викликатись при настанні такої події. Такий підхід до обробки натиснення кнопки є більш коректним, оскільки не витрачається час на перевірки стану кнопки в основному циклі програми.

Завдання

Зібрати схему з використанням зазначеної плати та RGB-світлодіоду та віддати її на перевірку викладачу. У випадку використання мезонінної плати (вставляється в роз'єм Arduino) рекомендується використовувати піни D9, D10, D11 для керування.

Увага! Завжди підключайте світлодіоди лише з використанням резисторів. Буде доступний модуль, на якому ці резистори вже є.

Повторне натиснення кнопки повинно повертати початковий ефект.

1. На початку світлодіод не показує нічого, потім він стає поступово максимально червоним, після чого яскравість червоного зменшується та світлодіод вимикається. Далі повторити для синього та зеленого. При натисненні на кнопку – від мінімальної яскравості – до максимальної (має бути білий колір).
2. Реалізувати перехід від жовтого кольору до фіолетового та навпаки. Натиснення кнопки – від червоного до синього.
3. Реалізувати перехід від червоного до зеленого, а потім до синього без вимкнення діоду та зациклити цю дію. При натисненні кнопки – перехід від вимкненого стану до жовтого кольору та навпаки.
4. Реалізувати перехід від білого до червоного, потім знову до білого, після чого цикл повторюється для синього та зеленого кольорів. При натисненні кнопки – теж саме, але початковий стан світлодіоду – вимкнений.

ЛАБОРАТОРНА РОБОТА № 4.
ПІДКЛЮЧЕННЯ КНОПКОВОГО БЛОКА ТА ОРГАНІЗАЦІЯ ЙОГО ОПИТУВАННЯ. ВИКОРИСТАННЯ ТАЙМЕРА

У цій роботі необхідно організувати опитування кнопкового блока та реалізувати реакцію на натиснення певної кнопки. Обробку натиснення кнопки слід реалізувати за допомогою функціональності таймера у платформі mbed. Рекомендована плата для виконання роботи – SMT32 Nucleo-L053R8.

Перед цим ми вже ознайомились з тим, як виконувати обробку однієї кнопки, проте часто пристрої можуть виконувати велику кількість функцій, кожній з яких відповідає кнопка. При цьому, варто враховувати, що ресурс входів/виходів у мікроконтролері не є безмежним, тому до їх використання також слід ставитися економно. Доволі поширеними є кнопкові модулі 4x4 (рис. 4.1), робота з якими передбачає використання 8 пінів (а не 16 відповідно до кількості кнопок).

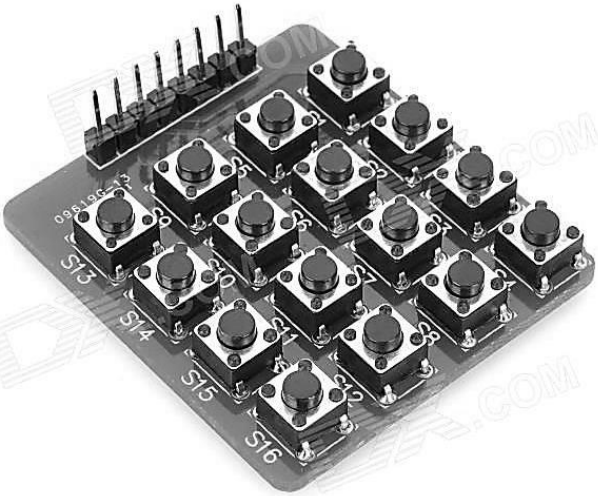


Рис. 4.1. Кнопковий модуль

Принцип організації опитування є доволі простим. У цьому модулі 8 пінів відповідають за 4 рядки та 4 стовпці. Подаючи по черзі сигнал 1 на входи рядків та зчитуючи стан входу (відносно мікроконтролера)

стовпців, можна визначити, яка кнопка була натиснута (припустимо, що є обмеження, що натиснутою може бути лише 1 кнопка). Наприклад, подаємо 1 на 1 лінію рядків та після цього зчитуємо 4 лінії стовпців, якщо в якомусь стовпці буде значення 1, то, значить, була натиснута клавіша. Далі цикл продовжується для 2,3 та 4 стовпців, після чого операція повторюється для 2 лінії рядків (1 лінія переводиться в 0) і т. ін.

Увага! Підключати кнопковий блок необхідно з використанням резисторів для того, щоб не спалити піни (хоча б підтягуючі резистори у складі мікроконтролера мають бути активовані)! Також не треба натискати одночасно декілька кнопок, оскільки це також може призвести до вигорання піну!

Для виклику функції через певні інтервали зазвичай використовують переривання таймера-лічильника. Однак платформа mbed пропонує обгортку навколо цієї концепції у вигляді класу Ticker. У результаті користувачу достатньо лише створити екземпляр цього класу та вказати функцію-callback, яка буде викликатись через заданий проміжок часу. Більш детальний контроль над таймерами пристрою доступний за допомогою бібліотеки HAL.

Для підключення матричної клавіатури будуть використані такі піни (рис. 4.2).

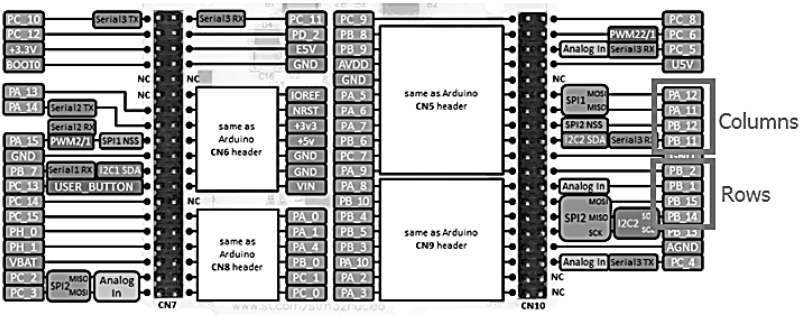


Рис. 4.2. Піни, що використовуються для підключення клавіатури

Розглянемо приклад (рис 4.3.), який пропонується для виконання завдання.

```
#include "mbed.h"

Ticker timer;
DigitalOut row0(PB_2);
DigitalOut row1(PB_1);
DigitalOut row2(PB_15);
DigitalOut row3(PB_14);
DigitalIn col0(PA_12, PullDown);
DigitalIn col1(PA_11, PullDown);
DigitalIn col2(PB_12, PullDown);
DigitalIn col3(PB_11, PullDown);

PwmOut pwm1(D6);
PwmOut pwm2(D5);
PwmOut pwm3(D4);

void set_outputs(uint8_t r0, uint8_t r1, uint8_t r2, uint8_t r3) {
    row0 = r0;
    row1 = r1;
    row2 = r2;
    row3 = r3;
}

uint8_t get_inputs() {
    if (col0 == 1)
        return 1;
    else if (col1 == 1)
        return 2;
    else if (col2 == 1)
        return 3;
    else if (col3 == 1)
        return 4;
    return 0;
}

void check_buttons(void) {
    uint8_t input;
    set_outputs(1, 0, 0, 0);
    input = get_inputs();
    if (input == 1) {
        if (pwm1 < 0.9)
            pwm1 = pwm1 + 0.1;
    } else if (input == 2) {
        if (pwm2 < 0.9)
            pwm2 = pwm2 + 0.1;
    } else if (input == 3) {
        if (pwm3 < 0.9)
            pwm3 = pwm3 + 0.1;
    }
    set_outputs(0, 1, 0, 0);
    input = get_inputs();
    if (input == 1) {
        if (pwm1 > 0.1)
            pwm1 = pwm1 - 0.1;
    } else if (input == 2) {
        if (pwm2 > 0.1)
            pwm2 = pwm2 - 0.1;
    } else if (input == 3) {
        if (pwm3 > 0.1)
            pwm3 = pwm3 - 0.1;
    }
    set_outputs(0, 0, 1, 0);
    input = get_inputs();
    set_outputs(0, 0, 0, 1);
    input = get_inputs();
    set_outputs(0, 0, 0, 0); // reset all outputs
}

int main()
{
    pwm1 = 0.0;
    pwm2 = 0.0;
    pwm3 = 0.0;
    timer.attach(&check_buttons, 0.1);
    while(1) {

    }
}
```

Рис. 4.3. Приклад програми

У наведеному прикладі за допомогою кнопок організується управління 3 каналами ШІМ, що підключені до RGB-світлодіоду. Можна збільшувати або зменшувати інтенсивність кожного з трьох каналів. Відповідно, для керування необхідно 6 кнопок. Їх обробка відбувається у check_buttons, яка викликається кожні 100 мс. Такий інтервал є цілком достатнім для того, щоб забезпечити коректну реакцію на дії користувача.

Завдання

1. Зіставити з кожною клавішею на матриці своє число у шістнадцятковому вигляді. Це число виводити на MAX7219.
2. Реалізувати 8 ефектів для RGB-діоду та зіставити їх з натисненням 8 кнопок.
3. Реалізувати рух точки (прямокутника) на модулі MAX7219.
4. Реалізувати «запуск» точки-снаряду на MAX7219: при натисненні на одну з 8 кнопок запускається точка, яка проходить шлях від лівого краю до правого та зникає. Номер кнопки визначає рядок, у якому з’являється точка.

ЛАБОРАТОРНА РОБОТА № 5.
ЗАСОБИ ВІДОБРАЖЕННЯ ІНФОРМАЦІЇ. ТЕКСТОВІ ДИСПЛЕЇ. КОНТРОЛЕР ДИСПЛЕЯ HD44780

У цій роботі необхідно дослідити протокол контролера дисплея HD44780 (маркування дисплея – 1602A). На основі нього організувати виведення інформації відповідно до завдання. Рекомендована плата – STM32 Nucleo L053R8.

Відображення інформації є однією з найважливіших загальних функцій для вбудованих систем. Для відображення найчастіше використовуються дисплеї. Існує велика кількість типів дисплеїв. Одними з найпростіших є текстові дисплеї, які призначені для виведення текстової інформації та псевдографіки на основі тексту. Вони використовуються там, де достатньо виведення текстових повідомлень про стан системи, показники датчиків тощо. Одним з найбільш поширених контролерів текстових дисплеїв є HD44780.

Керування цим пристроєм є більш складним, ніж попередньо розглянутими пристроями з точки зору використання виводів. Необхідно 8 пінів для передачі даних, а також 2 піни для контролю за керівними лініями (Register Select та Enable), які показані на рис. 5.1. Живлення та землю для дисплея слід підключити безпосередньо з плати.

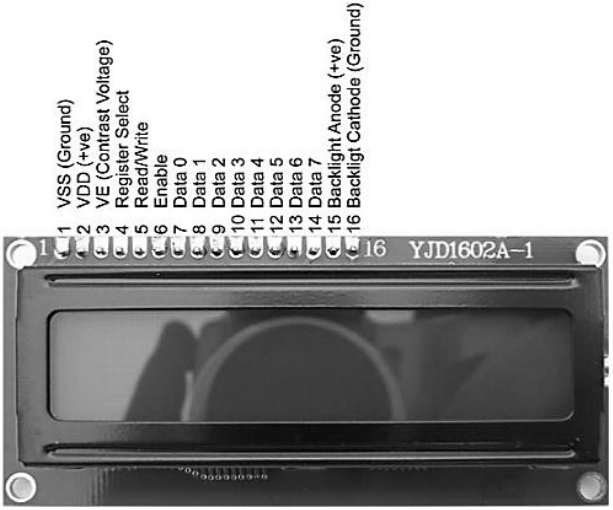


Рис. 5.1. Функціональне призначення виводів дисплея

Дисплей може працювати як з використанням 4-бітної шини даних, так і 8-бітної. У роботі буде використовуватись 8-бітна шина, оскільки вона не потребує додаткової обробки даних для запису (запис символів, рядків не потребуватиме їх додаткових обробок). Вивід VE (V0) підключають до живлення через змінний резистор, за допомогою якого регулюють контрасність символів на дисплеї. Виводи A (анод) та K (катод) варто підключити до живлення та землі відповідно. Вони відповідають за підсвічування. Вхід RW підключають до землі. Приклад підключення для Arduino показаний на рис. 5.2.

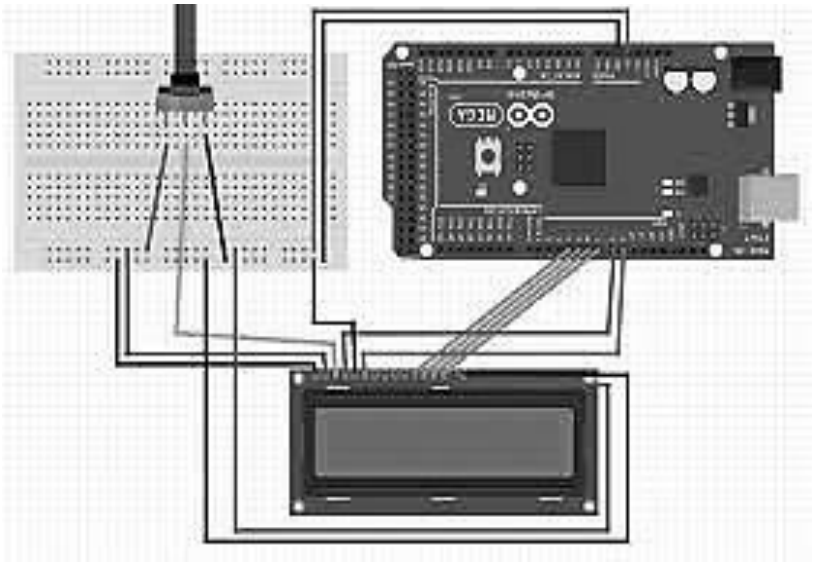


Рис. 5.2. Підключення дисплея
з використанням 4-бітової шини даних

Робота з дисплеєм полягає в послідовності передачі команд та даних. За допомогою команд ініціалізується дисплей та встановлюється адреса для запису, а також виконується очищення екрана. Розглянемо це детальніше на наступному прикладі (рис 5.3).

```
#include "mbed.h"

DigitalOut rs(D12);
DigitalOut e(D11);
DigitalOut d0(D2);
DigitalOut d1(D3);
DigitalOut d2(D4);
DigitalOut d3(D5);
DigitalOut d4(D6);
DigitalOut d5(D7);
DigitalOut d6(D8);
DigitalOut d7(D9);

void write_data_to_port(uint8_t data) {
    d0 = ((data >> 0) & 0x01);
    d1 = ((data >> 1) & 0x01);
    d2 = ((data >> 2) & 0x01);
    d3 = ((data >> 3) & 0x01);
    d4 = ((data >> 4) & 0x01);
    d5 = ((data >> 5) & 0x01);
    d6 = ((data >> 6) & 0x01);
    d7 = ((data >> 7) & 0x01);
}

void send_data(uint8_t data) {
    rs = 1;
    write_data_to_port(data);
    e = 1;
    wait(0.04f);
    e = 0;
}

void send_command(uint8_t command) {
    rs = 0;
    write_data_to_port(command);
    e = 1;
    wait(0.04f);
    e = 0;
}

void init_lcd(void) {
    wait(0.02f);
    send_command(0b00110000);
    send_command(0b00110000);
    send_command(0b00110000);
    send_command(0b00111000);
    send_command(0b00001111);
    send_command(0b00000001);
    send_command(0b00000110);
    send_command(0b00000010);
}

void lcd_set_address(uint8_t address) {
    send_command(0b10000000 | address);
}

void write_string(char * str) {
    do {
        send_data(*str);
    } while(*++str);
}

int main() {
    init_lcd();
    send_data('a');
    write_string("foo");
    while(1) {

    }
}
```

Рис 5.3. Приклад програми

Перш за все, зверніть увагу на те, які піни використовуються для керування дисплеєм та намагайтеся слідувати підключенню, яке наведено в прикладі (рис. 5.4).

Далі більш детально зупинимось на реалізованих функціях. Розрізняють передачу команд і передачу даних. Вони відрізняються станом лінії RS. Для ініціалізації необхідно передати декілька констант як команди на початку. Після ініціалізації дисплея, на нього можна виводити інформацію. Попередньо, за необхідності, встановлюється адреса, у яку виводиться символ. Дисплей може відображати 2 рядки по 16 символів. Адреса першого рядка у двійковій формі – 0x10000000, для того, щоб записувати у другий рядок до цього значення необхідно додати 40. Для очистки екрана необхідно передати команду з кодом 0x01.

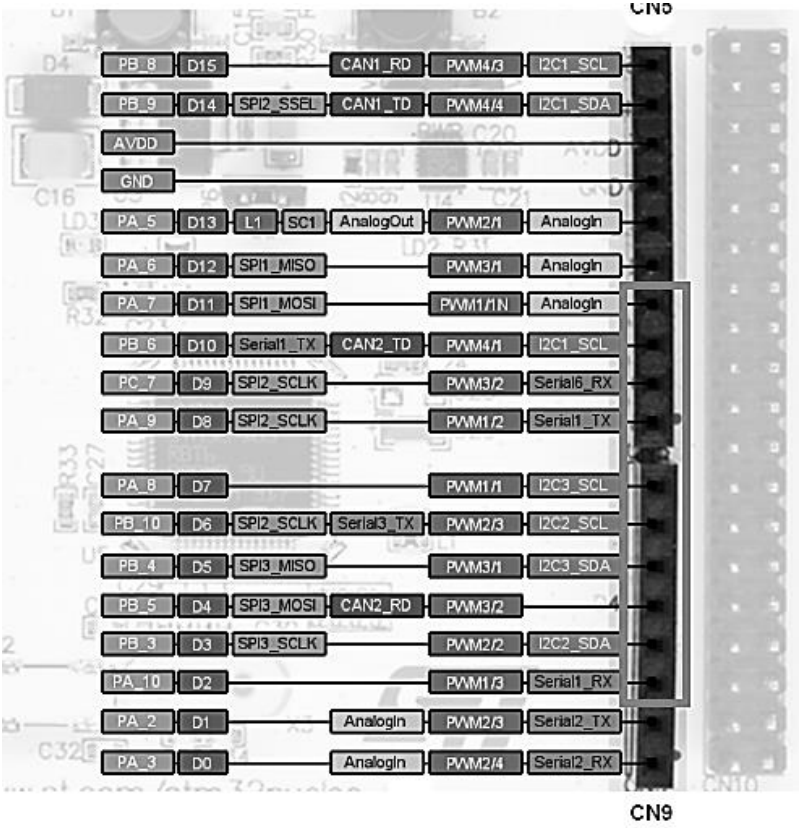


Рис. 5.4. Піни, що рекомендуються для підключення дисплея

Завдання

Організувати підключення дисплея за допомогою bread-board. Безпосередньо перед підключенням надати на перевірку викладачеві. У випадку, якщо підключення коректне, то подати напругу та перевірити, що у верхньому рядку відображаються квадрати. Відрегулювати яскравість за допомогою потенціометра. Спробувати вивести символ/рядок на дисплей за допомогою функцій з прикладу. Переконатись, що функції працюють. Виконати завдання відповідно до обраного варіанта.

1. Вивести ім’я в першому рядку, після чого воно через певний час переміщується у кінець та зникає, з’являючись у нижньому рядку, і так само проходить до кінця.

2. Вивести рядок з числами (0–F) наступним чином: спочатку з'являється 0, потім 1 і т. ін. Після того, як рядок заповнений, 0 зникає, через деякий час зникає 1 і т. ін. Повторити для 2 рядка.

3. Одночасно в 1 та 2 рядку є анімація слова. На початку слова розташовані на різних кінцях та починають зближуватись, переходять до протилежного кінця. Слова однакової довжини.

4. Те саме, що для 3, але слова різної довжини. Кожен цикл має починатися одночасно для обох слів.

ЛАБОРАТОРНА РОБОТА № 6.
УПРАВЛІННЯ СЕРВОДВИГУНАМИ

У цій роботі необхідно реалізувати керування сервомотором відповідно до завдання. Рекомендована плата для виконання завдання – STM32 Nucleo L053R8.

Одним з видів навантаження, яким можна керувати за допомогою ШІМ є двигуни і, зокрема, серводвигуни. У роботі необхідно працювати з серводвигуном Tower Pro SG90. Для роботи з ним необхідно підключити 3 піни (рис. 6.1.):

– Коричневий – для землі;

– Червоний – для живлення (відповідно до документації може працювати в діапазоні 3–7.5 В; для роботи рекомендується підключати до 5 В);

– Жовтий – для керування (підключається сигнал ШІМ).

Рис. 6.1. Зовнішній вигляд сервоприводу

Положення валу двигуна визначається параметрами ШІМ, які подаються на вхід. Для керування двигуном необхідно налаштувати частоту ШІМ у 50 Гц або період 0.02 с. Коефіцієнт заповнення при цьому має становити приблизно від 0.025 (0°) до 0.125 (180°), як показано на рис. 6.2.

У прикладі, що буде розглянутий як базовий, будуть використані можливості бібліотеки HAL для більш гнучкого налаштування таймера при генерації ШІМ. Імпортувати програму-приклад можна за адресою https://os.mbed.com/users/codebreaker7/code/LAB_Servo/ (для імпорту

натиснути кнопку Import, після чого натиснути посилання у напису Click here to import from URL та вставити вказане посилання).

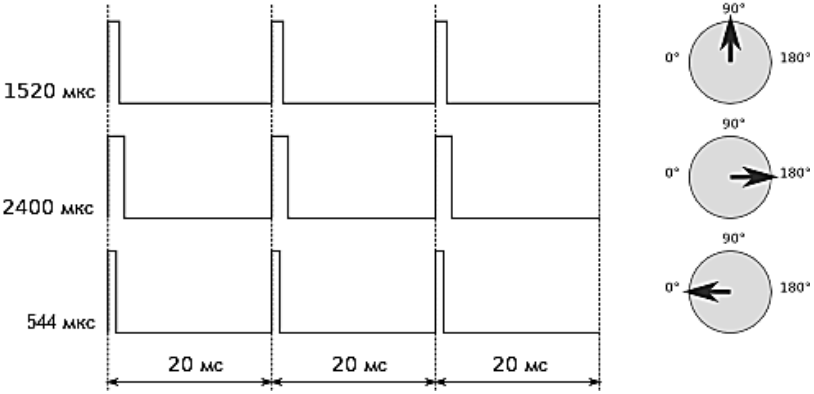


Рис. 6.2. Параметри ШІМ для керування сервоприводом

Для налаштування ШІМ за допомогою HAL необхідно створити 2 екземпляри структур типу TIM_HandleTypeDef та TIM_OC_InitTypeDef. Перша з них використовується для того, щоб вказати загальні налаштування роботи таймера, а друга – для налаштування каналу ШІМ. За замовчанням mbed ініціалізує пристрій таким чином, щоб він працював на максимальній частоті. У нашому випадку це значення – 32 МГц. Для роботи з сервоприводом необхідно, щоб частота ШІМ становила 50 Гц. Саме для цього ми розраховуємо значення поля Prescaler як 9 (10–1), а значення поля Period – 64000 – 1. Такі налаштування дозволяють отримати подію оновлення таймера кожні 0.02 с, або з частотою 50 Гц:

$$32\ 000\ 000 / (10 * 64\ 000) = 50.$$

Після цього нам необхідно задати налаштування для коефіцієнта заповнення ШІМ. У прикладі це значення розраховано приблизно. На початку він становить

$$64\ 000 * 0.025 = 1600,$$

а коли доходить до крайнього положення – 8 333.

Завдання

Підключити сервоприводи до плати та запустити приклад. Двигуни з'єднані за допомогою пластикових деталей, які надають можливість промоделювати роботу, наприклад, маніпулятора. Перевірити їхню

роботу. Реалізувати управління серводвигуном відповідно до завдання. При підключенні використовувати піни PA0 та PA1 для подачі сигналу ШІМ, один із двигунів підключити до напруги 5 В, а інший – 3.3 В. При виконанні завдання пам’ятайте, що кут обертання змінюється від 0 до 180°.

1. Реалізувати симуляцію того, що маніпулятор бере з одного місця предмет і переносить його на інше і т. ін.
2. Реалізувати симуляцію того, що маніпулятор під час руху малює коло.
3. Маніпулятор проходить усі крайні точки по діагоналі.
4. Маніпулятор починає рух у точці 0 піднімається дугою і опускається в положенні 90 градусів, щоб підняти предмет. Після цього він переносить його у позицію 180°.

ЛАБОРАТОРНА РОБОТА № 7.
УНІПОЛЯРНІ КРОКОВІ ДВИГУНИ

Крокові двигуни є надзвичайно поширеними у різноманітних механічних системах. Вони називаються кроковими тому, що переміщення забезпечується за допомогою виконання серії кроків – маленьких мінімальних переміщень, для виконання яких на двигун (а точніше, на драйвер двигуна) необхідно подати послідовність команд. Одна така послідовність забезпечує обертання вала на певну мінімальну позицію, наприклад, 1/200 кола. Відповідно, необхідно подати 200 таких послідовностей для того, щоб вал зробив повний оберт.

У цій роботі необхідно організувати роботу уніполярного крокового двигуна 28-BYJ48 та драйвера ULN2003. Цей двигун виконує оберт за 64 кроки. Варто зазначити, що точність позиціонування для цього двигуна є доволі низькою. Він здатен робити 100 кроків за 1 секунду, відповідно, швидкість обертання є меншою за 2 оберти на секунду.

На драйвері доступні 6 виводів для підключення зі сторони відлагоджувальної плати (включаючи живлення та землю). Також доступний роз'єм для підключення крокового двигуна (рис. 7.1).

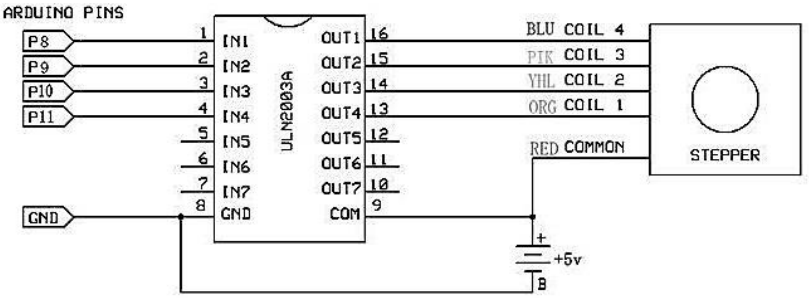


Рис. 7.1. Схема підключення

Для того, щоб двигун почав робити кроки, необхідно подати певну послідовність команд з частотою до 100 Гц. У цьому випадку для обертання за годинниковою стрілкою послідовність має бути такою, як показано на рис. 7.2.

SWITCHING SEQUENCE

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

Рис. 7.2. Послідовність команд для виконання кроку

Не забувайте додавати часові проміжки (наприклад, 3 мс) між виконанням кроків, оскільки сигнали з частотою більше за вказану не будуть сприйматись і обертання не відбуватиметься. Для обертання у зворотному напрямку послідовність команд є зворотною.

Завдання

Підключити крововий двигун, драйвер та відлагоджувальну плату. Живлення для двигуна забезпечується з плати. Запустити крововий двигун відповідно до опису вище. Організувати обертання двигуна відповідно до завдання.

1. Двигун робить 2 оберти вперед та 2 оберти у зворотному напрямку. При натисненні на кнопку кількість обертів – 1.
2. Двигун робить 1 оберт вперед, 1 секунду стоїть, після чого виконує один оберт назад і т. ін. При натисненні на кнопку – зворотна послідовність.
3. Двигун робить 0.5 оберта вперед та 1 оберт у зворотному напрямку. При натисненні на кнопку кількість обертів – навпаки.
4. Двигун за замовчанням стоїть, необхідно забезпечити наступне керування за допомогою кнопки. Перше натиснення на кнопку – двигун обертається 1 раз. Друге – оберт у зворотному напрямку. Третє – двигун обертається 2 рази. Четверте – 2 рази у зворотному напрямку. Так відбувається до кількості обертів – 5, після чого знову має бути 1 оберт.

ЛАБОРАТОРНА РОБОТА № 8.
РОБОТА З ДИСПЛЕЄМ З ВИКОРИСТАННЯМ ІНТЕРФЕЙСУ SPI

В одній з попередніх робіт Ви ознайомились з тим, як працювати з символьним дисплеєм. Тим не менш, набагато більш поширеними на сьогодні є графічні дисплеї, які дозволяють відображати більшу кількість інформації та надають більше можливостей. У цій роботі необхідно ознайомитись з дисплеєм на основі контролера ILI9163, провести його ініціалізацію та реалізувати певний мінімальний функціонал для роботи з дисплеєм.

Дисплей керується за допомогою вже знайомого інтерфейсу SPI. При цьому у контролера з'являється ще 1 вхід. Він визначає, що саме подається на вхід: команди або дані. Також наявний вхід Reset, який має бути встановлений в 1.

На рис. 8.1 показано зовнішній вигляд дисплея.

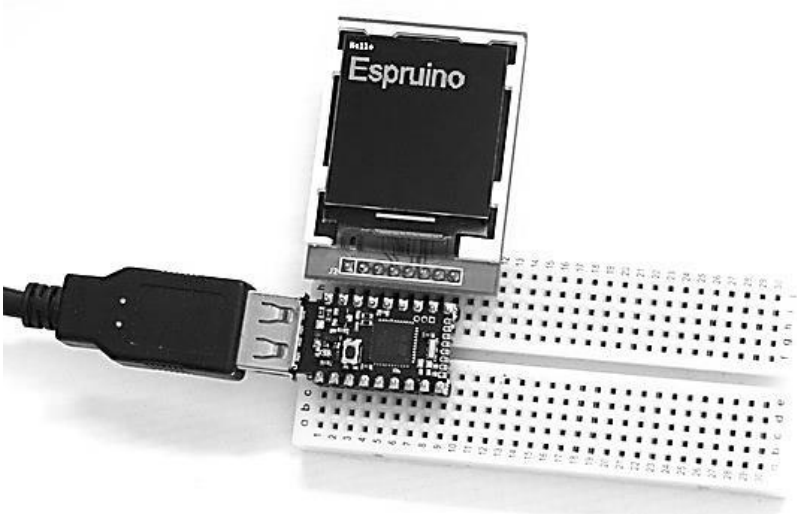


Рис. 8.1. Зовнішній вигляд модуля дисплея

Для початку необхідно підключити дисплей до плати. Рекомендовані піни для підключення можна переглянути у проекті-прикладі (імпортувати проект можна звідси – https://os.mbed.com/users/codebreaker7/code/LAB_DISPLAY/).

Для того, щоб почати працювати безпосередньо з дисплеєм, на початку треба подати послідовність команд, які виконають його ініціалізацію. Лише після цього він буде готовий до відображення інформації. У проекті-прикладі код ініціалізації винесено в окремий заголовковий файл. Зверніть увагу на вміст цього файлу та те, як він використовується за допомогою директиви #include безпосередньо у функції main. Загалом, можна зазначити, що робота з дисплеєм складається з того, що контролеру дисплея відправляються або команди, або дані (за це відповідає пін AO: якщо слід відправити команду, то його значення – 0, дані – 1). При ініціалізації дисплея використовуються не лише команди, а і дані, для того, щоб встановити параметри роботи дисплея.

Після того як дисплей проініціалізовано, на ньому можна побачити зображення типу білий шум (у перший раз за умови, що достатньо довго не було живлення і, що не виводилось ніяких зображень). Після цього можна переходити безпосередньо до того, щоб керувати тим, що відображається на екрані. Розмір екрана становить 128x128 пікселів, а для представлення 1 пікселя використовуються 2 байти (формат RGB565). Відповідно, уся пам’ять зображень у такому випадку займатиме понад 32 кб. Контролер дисплея відповідає за те, щоб виводити зображення з області графічної пам’яті на дисплей. Для того, щоб записати дані у область графічної пам’яті необхідно для початку встановити вікно – область, у яку відбуватиметься запис, – а після цього перейти безпосередньо до запису даних у цю область пам’яті. За встановлення вікна у прикладі відповідає функція setAddress, а за відправку даних – sendColorData. Зверніть увагу на те, що розмір масиву, який використовується для передачі даних про колір – 1024 байт, що у 32 рази менше за розмір, яку займає область графічної пам’яті дисплея. Це зроблено для того, щоб зекономити ресурси пристрою, оскільки розмір оперативної пам’яті SRAM становить лише 8 кб. Тому для збереження в пам’яті всього зображення на мікроконтролері просто не вистачає місця. Тож, рекомендується використовувати такий підхід:

1. Підготувати масив, заповнивши його необхідними даними.
2. Встановити адресу для виведення.
3. Оновити відповідну частину екрана, передавши підготовлені дані.

Такий принцип необхідно буде реалізувати відповідно до завдання.

Завдання

Підключити дисплей до плати, запустити програму-приклад та перевірити роботу дисплея. Його зручно підключати за допомогою макетної плати. Живлення та землю можна подавати безпосередньо з плати. На піни VCC, LED, RST слід подати лог. 1. GND – підключити до піна GND на платі. Інші піни підключаються відповідно до налаштувань SPI.

1. Реалізувати прохід зеленої «змійки» по діагоналі. Вона з’являється та зникає через деякий час.
2. Реалізувати обхід червоною «змійкою» периметра дисплея.
3. Реалізувати заповнення дисплея різнокольоровими квадратами з подальшим їх зникненням.
4. Реалізувати виведення на дисплей 3 будь-яких чисел або літер.

ЛАБОРАТОРНА РОБОТА № 9.
РОБОТА З ДРАЙВЕРОМ ДВИГУНІВ L298

Драйвер двигунів L298 є пристроєм для керування двигунами. Він має силову частину та частину, що призначена для організації логіки керування. У цій роботі необхідно організувати керування 2 двигунами, що забезпечують рух платформи. На платформі, окрім самого модуля драйвера, є 2 колеса (додатково 1 – для стабілізації платформи), блок живлення, DC-DC-перетворювач, а також встановлюється плата для керування.

На рис. 9.1 показано зовнішній модуль драйвера.

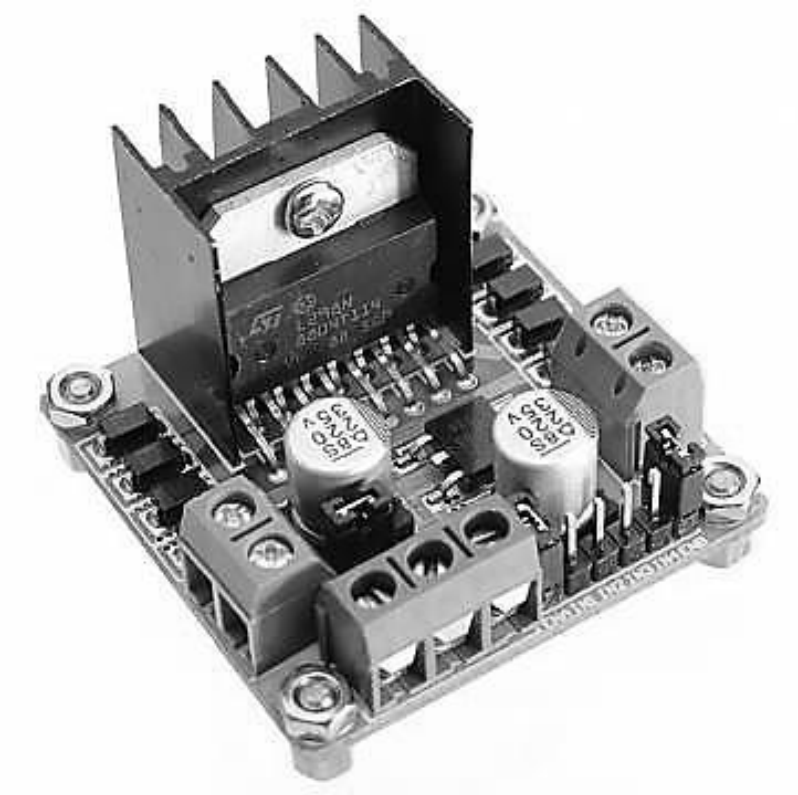


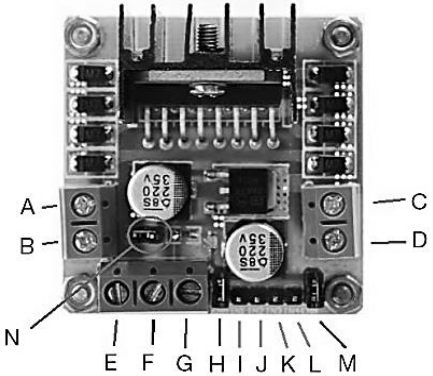
Рис. 9.1. Зовнішній вигляд модуля драйвера

Під час виконання роботи варто уважно слідкувати за модулем DC-DC-перетворювача, оскільки на ньому виставлена достатня вихідна напруга, а поворот змінного резистора на цьому модулі може призвести до того, що схема не буде працювати без видимих причин.

У платформі використовуються мотори, які показані на рис. 9.2.

Рис. 9.2. Мотор для робоплатформи

Цей мотор є достатньо потужним для того, щоб рухати таку масивну платформу. Керування двигуном здійснюється надзвичайно просто: подачею необхідних логічних рівнів на входи драйвера L298. Для керування 1 двигуном використовується пара входів. Функціональне призначення виходів та входів модуля представлено на рис. 9.3.



Part	Name	Type
A	A+	Output
B	A-	
C	B-	
D	B+	
E	Voltage Supply (Motor)	Power
F	Gnd	
G	5V	
H	Motor A PWM	Control
I	In 1	Input
J	In 2	
K	In 3	
L	In 4	
M	Motor B PWM	Control
N	5V regulator Enabler	Jumper

Рис. 9.3. Функціональне призначення виходів та входів модуля драйвера двигунів

Входи для керування двигунами позначені жовтогарячим кольором (I-L). Подача 1 та 0 означає обертання вперед, навпаки – у зворотному напрямку.

Завдання

Налаштувати платформу для керування двигунами: вставити елементи живлення у спеціальний відсік та запрограмувати плату, на якій буде проводитись тестування і підключити її до входів керування. Подати живлення на драйвер (за допомогою кнопки) та продемонструвати роботу у 3 режимах:

1. Рух вперед.
2. Рух назад.
3. Поворот ліворуч та праворуч.

Підлаштувавши рівні тривалості сигналів, провести демонстрацію безпосередньо біля комп'ютера (плата буде живитись від USB). Слідкуйте за тим, щоб платформа не падала з великої висоти!

ЛАБОРАТОРНА РОБОТА № 10.
ВІДОБРАЖЕННЯ ІНФОРМАЦІЇ НА OLED-ДИСПЛЕЇ

Мета роботи – навчитись виводити графічну інформацію (текстову та у вигляді зображень) на OLED-дисплей з контролером SSD1306.

Дисплей, який використовується у цій роботі належить до дисплеїв, виконаних за технологією OLED. Це дозволяє значно зменшити споживання порівняно з TFT-дисплеями. Тому такі дисплеї підходять для систем, у яких необхідно забезпечити якомога менше споживання. Конкретний дисплей здатен відображати інформацію за допомогою переключення пікселів у білий колір. Розмір такого дисплея у пікселях становить 128x64, а розмір діагоналі – 0,96 дюйма. За відображення графічної інформації на дисплеї відповідає контролер SSD1306. З точки зору доступу до цього контролера, команди та дані передаються за допомогою інтерфейсу I2C. Живлення до модуля дисплея подається в діапазоні 3.3-5 В.

Для спрощення роботи з дисплеєм доцільно скористатись бібліотекою для роботи з ним. Для цього слід імпортувати файли бібліотеки у проект. У репозиторіях mbed зберігаються, окрім користувацьких проектів, і бібліотеки, які можуть бути підключені до проекту. Для того, щоб імпортувати бібліотеку, слід натиснути кнопку Import на панелі меню та у вікні, яке відкриється, перейти на вкладку Libraries. У полі пошуку ввести “SSD1306”. У результаті має з’явитись результат пошуку (рис. 10.1). Насправді, для такого дисплея доступна велика кількість бібліотек. Одні з них є широко відомими, інші – бібліотеки адаптовані під конкретні проекти. Оптимальним варіантом є вибір бібліотеки з найбільшою кількістю підключень, тому рекомендується використовувати бібліотеку Adafruit_GFX. Для того, щоб імпортувати бібліотеку до свого проекту, слід натиснути кнопку Import у правому верхньому куті.

Рис. 10.1. Результати пошуку бібліотеки для дисплея

Бібліотека підключається до проекту в вигляді вихідних файлів. Зміну у структурі проекту можна спостерігати у дереві проектів (рис. 10.2).

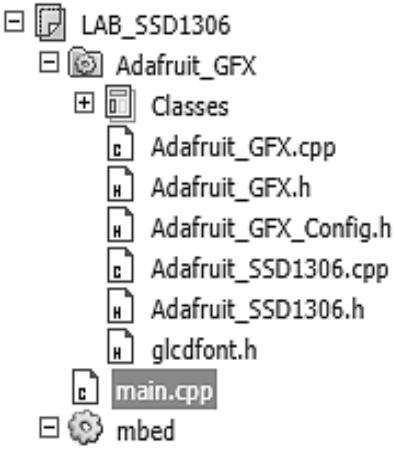


Рис. 10.2. Проект з підключеною бібліотекою Adafruit_GFX

Для того, щоб запустити найпростіший приклад та почати використовувати бібліотеку достатньо провести ініціалізацію модуля I2C на платі (рекомендується використовувати піни D14, D15 для цього) і провести ініціалізацію екземпляра класу для роботи з графікою – Adafruit_SSD1306_I2c (рис. 10.3).

```
#include "mbed.h"
#include "Adafruit_GFX.h"
#include "Adafruit_SSD1306.h"

I2C i2c(D14, D15);
Adafruit_SSD1306_I2c disp(i2c, A0, SSD_I2C_ADDRESS, 64, 128);

int main() {
    while(1) {

    }
}
```

Рис. 10.3. Базова програма для роботи з дисплеєм

У результаті запуску програми на дисплеї має з'явитись зображення, яке є логотипом Adafruit, а також напис «Adafruit Industries» (рис. 10.4).

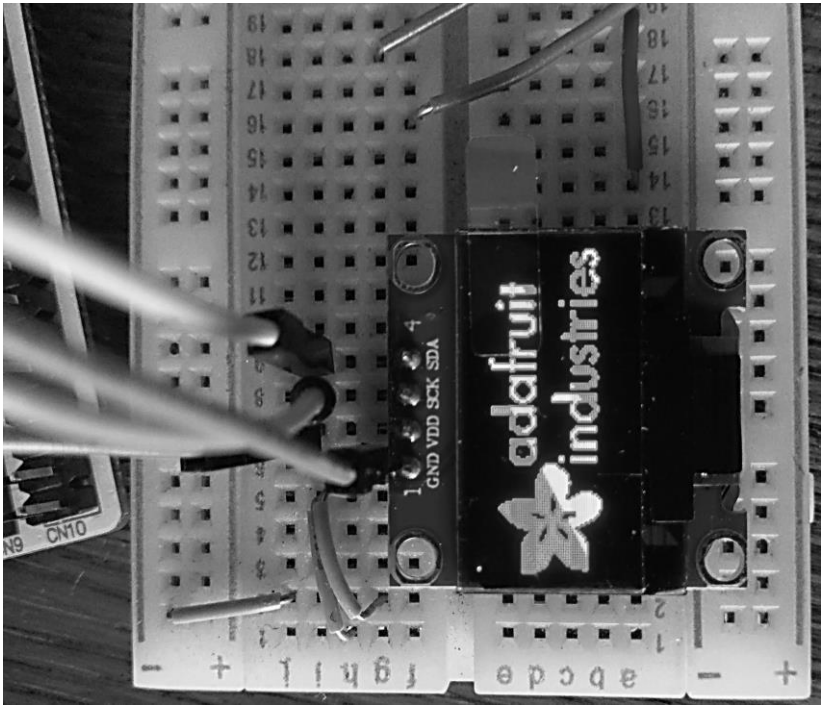


Рис. 10.4. Вигляд дисплея після запуску прикладу

Варто також розуміти, що підключена бібліотека дозволяє значно спростити розробку для обраного дисплея, проте забирає при цьому доволі багато ресурсів. Наприклад, для плати STM32L053 використання пам'яті в найпростішому випадку перевищує 50 % відсотків ресурсів пам'яті мікроконтролера (рис. 10.5).

Бібліотека пропонує розробнику набір функцій для виведення графіки на дисплей. Розглянемо детальніше основні функції, які наявні у бібліотеці:

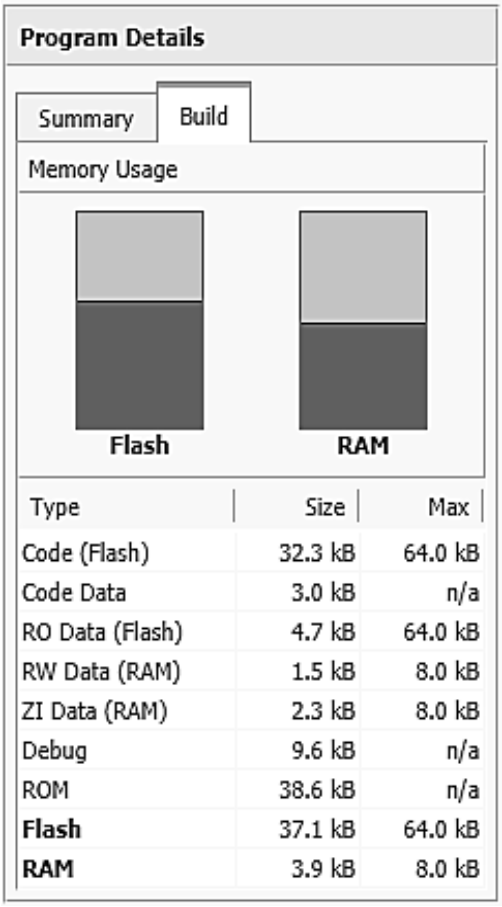
– `clearDisplay` – очищує буфер дисплея;

– `display` – виводить на дисплей вміст буфера;

– `fillScreen` – заповнює буфер кольором (варіанти усього 2 – або чорний, або білий);

– `drawChar` – виводить у буфер зображення символу у відповідній позиції у буфері;

– `drawPixel` – встановлює значення пікселя в екранному буфері.



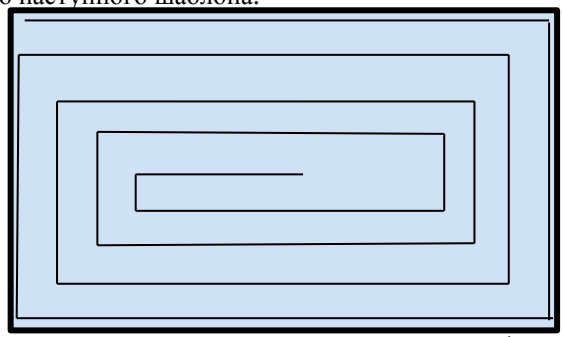
Type	Size	Max
Code (Flash)	32.3 kB	64.0 kB
Code Data	3.0 kB	n/a
RO Data (Flash)	4.7 kB	64.0 kB
RW Data (RAM)	1.5 kB	8.0 kB
ZI Data (RAM)	2.3 kB	8.0 kB
Debug	9.6 kB	n/a
ROM	38.6 kB	n/a
Flash	37.1 kB	64.0 kB
RAM	3.9 kB	8.0 kB

Рис. 10.5. Використання пам'яті програмою-прикладом

Також наявні функції для малювання графічних примітивів. Як зрозуміло з опису функцій, робота з виведення зображення організована через буфер. У той момент, коли у буфері сформоване зображення для виведення, має бути викликана функція display. Характерною є підтримка виводу рядків, оскільки текстова інформація є найбільш типовою для представлення таких датчиків та ін. Шрифти описані в окремих файлах і підтримуються три різні розміри символів. Також завдяки функції drawBitmap є підтримка виведення зображень на дисплей. Вони мають бути відформатовані у вигляді масиву.

Завдання

1. Реалізувати ефект переміщення прізвища периметром дисплея.
2. Реалізувати ефект поступового заповнення дисплея лініями спочатку з одного кута, а потім – з іншого.
3. Реалізувати ефект руху змійки від периметра до центру відповідно до наступного шаблона.



4. Поступове заповнення дисплея квадратами, потім колами від периметра до центру через одного.
5. Вивести на екран зображення за допомогою функції drawBitmap та запрограмувати його довільний рух дисплеєм.

ЛАБОРАТОРНА РОБОТА № 11.
ДАТЧИК ТЕМПЕРАТУРИ ТА ВОЛОГОСТІ DHT11

Мета роботи – навчитись використовувати датчик температури та вологості DHT11 для вимірювання відповідних параметрів з виведенням отриманих даних на дисплей.

Датчик температури та вологості DHT11 дозволяє вимірювати одночасно два показники: температуру та вологість. Діапазон вимірювання температури вказаний як 0...50°C, а діапазон значень вологості – 20–90 % (хоча останній залежить від температури). Датчик забезпечує відносно невелику похибку, тому його можна використовувати в системах, де точністю вимірювань можна знехтувати. Допустиме живлення для подачі на датчик становить 3–5.5 В, тому з ним можуть працювати мікроконтролери, які працюють лише з логікою 3.3 В. Інтерфейс передачі даних у дечому схожий на інтерфейс UART, проте реалізувати його безпосередньо з використанням модуля UART є доволі проблематичним. Це пов'язано з тим, що необхідно перемикатись між станами піну (він виконує функції і входу, і виходу залежно від фази передачі даних). Взаємодія мікроконтролера з датчиком відбувається за допомогою одно-провідного (one-wire) інтерфейсу. Для підключення датчика може знадобитись додатковий резистор, оскільки лінія передачі даних має бути підтягнута до рівня логічної одиниці за допомогою резистора з опором 10 кОм. У випадку використання модуля з датчиком на платі такий резистор не потрібний, оскільки він вже наявний на модулі плати. DHT11 має корпус синього кольору з чотирма пінами, проте для роботи з датчиком достатньо лише трьох (двох – для подачі живлення і третій – для передачі даних). Оскільки один з виводів не використовується, то у датчиках, які змонтовані на плати, наявні лише три виводи.

> ***Увага!*** У випадку використання таких модулів для виконання лабораторної роботи слід перевірити призначення пінів у модулі для того, щоб їх коректно підключити. Некоректне підключення може спричинити вихід датчика з ладу.

Для роботи з датчиком можна скористатись готовими бібліотеками або програмами, які наявні в репозиторії. Вони мають готові реалізації функцій із ініціалізації та зчитування даних з датчика, виклики яких необхідно лише адаптувати під обрану платформу. У цьому прикладі проведемо імпорт програми, а не бібліотеки. Для цього при натисненні кнопки Import слід перейти на вкладку Programs. Після цього слід ввести ім'я програми – Nucleo_DHT11_Example (рис. 11.1.). Програму можна також імпортувати за допомогою URL: https://os.mbed.com/users/ledonger/code/Nucleo_DHT11_Example/.

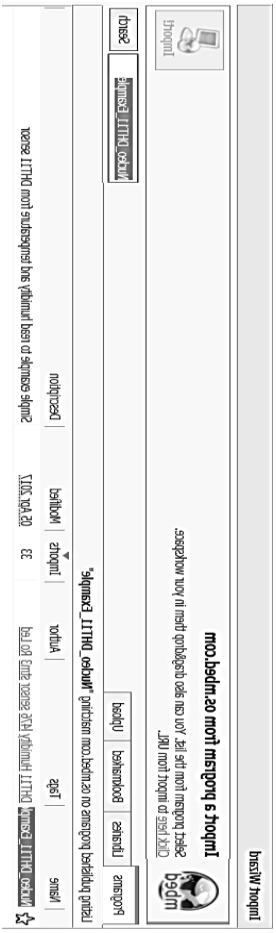


Рис. 11.1. Імпорт програми для реалізації прикладу

У заголовковому файлі (рис 11.2.) імпортованої програми міститься повний опис класу для роботи з DHT11. Найбільш важливою в ньому є опис функції `readData`.

```
int readData(void){
    this->data = 0;
    this->dataPin.output();
    // Request a measurement (low during t > 18ms)
    this->dataPin = 0;
    wait_ms(20);
    this->dataPin = 1;

    //Wait for the sensor to take control and set low level /!\
    wait_us(20);

    this->dataPin.input();

//TODO Check if timing is correct (low : 80µs ; high 80µs)
    // Wait until end of 80µs low
    while(!this->dataPin.read());
    // Wait until end of 80 µs high
    while(this->dataPin.read());

    // Sensor reply 40bits
    for(iBit=0; iBit<40; iBit++) {
        this->data = this->data << 1; // Shift for new number
        this->timer.stop();
        this->timer.reset();

        // Wait for low level to end
        while(!this->dataPin.read());
        this->timer.start();
        // Count time while high level
        while(this->dataPin.read());

        if(this->timer.read_us() > 50)
        {
            //This bit is '1'
            this->data++;
        }
    }

    wait_ms(250);

    //Checking checksum
    if((this->data & 0x00000000000000ff) != ((this->data & 0x000000ff00000000) >> 32) +
                                            ((this->data & 0x00000000ff000000) >> 24) +
                                            ((this->data & 0x0000000000ff0000) >> 16) +
                                            ((this->data & 0x000000000000ff00) >> 8))
    {
        return -1;
    }

    return 1;
}
```

Рис 11.2. Вміст заголовкового файлу

Слід звернути увагу на переключення стану піну з виходу на вхід з подальшим зчитуванням даних. Зчитування відбувається через задані часові проміжки, для вимірювання яких використовується таймер.

Розглянемо детальніше функцію main, яка використовується у прикладі (рис. 11.3). На початку оголошено основні змінні для збереження даних та контролю за процесом виведення даних. Серед визначених змінних є рядок на двадцять символів для виведення повідомлень на дисплей. Спочатку на дисплеї має з'явитись стандартне повідомлення (виводиться у момент створення відповідного об'єкта) для того, щоб можна було перевірити, чи коректно підключений дисплей. Далі в циклі з інтервалом 2 с з датчика зчитуються дані. Зчитані дані конвертуються в рядок та виводяться на дисплей.

```
int main() {
    int s;
    int humidity, temp;
    char str[20];
    wait(2);
    disp.clearDisplay();
    while(1) {
        s = sensor.readData();
        if (s == -1) {
            disp.drawPixel(10, 10, WHITE);
            sprintf(str, "No data\n");
            drawString(disp, str, 20, 20);
        } else {
            disp.clearDisplay();
            temp = sensor.getTemperature();
            humidity = sensor.getHumidity();
            sprintf(str, "%d\n", temp);
            drawString(disp, str, 20, 20);
            sprintf(str, "%d\n", humidity);
            drawString(disp, str, 20, 40);
        }
        disp.display();
        wait(2);
    }
}
```

Рис. 11.3. Реалізація функції main

У ході виконання цієї роботи пропонується також використовувати OLED-дисплей на базі контролера SSD1306 з попередньої роботи. Підключивши дисплей, можна побачити, що дані виводять на нього (рис. 11.4). Для підключення датчика, окрім пінів живлення та землі, слід виокремити один пін для роботи з даними.

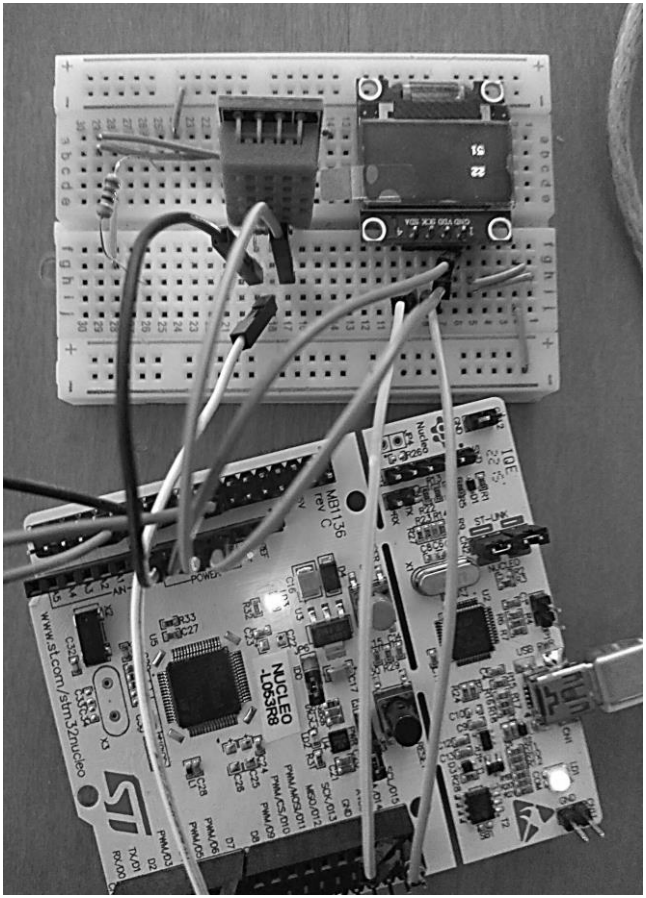


Рис. 11.4. Перевірка роботи прикладу на макетній платі

Завдання

Запустити програму-приклад, забезпечити більш інформативний вивід повідомлень на дисплей, переконатись у працездатності датчика, змінюючи температуру.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ultra-low-power 32-bit MCU Arm®-based Cortex®-M0+ [Електронний ресурс]. – Режим доступу : https://www.st.com/resource/en/ datasheet/stm32l053r8.pdf. – Загол. з екрана.
2. STM32 Nucleo-64 development board with STM32L053R8 MCU [Електронний ресурс]. – Режим доступу : https://www.st.com/en/ evaluation-tools/nucleo-l053r8.html. – Загол. з екрана.
3. Mbed OS [Електронний ресурс]. – Режим доступу : https://www.mbed.com/en/platform/mbed-os/. – Загол. з екрана.
4. SSD1306 [Електронний ресурс]. – Режим доступу : https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf. – Загол. з екрана.
5. L298 Dual Full-Bridge Driver [Електронний ресурс]. – Режим доступу : https://www.sparkfun.com/datasheets/Robotics/ L298_H_Bridge. pdf. – Загол. з екрана.
6. DHT11 Humidity & Temperature Sensor [Електронний ресурс]. – Режим доступу : https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. – Загол. з екрана.
7. LCD Module Product Specification [Електронний ресурс]. – Режим доступу : https://www.displaytech-us.com/sites/default/files/display-data-sheet/DT018ATFT-v10.pdf. – Загол. з екрана.
8. HD44780U (LCD-II) [Електронний ресурс]. – Режим доступу : URL : https://www.sparkfun.com/datasheets/LCD/HD44780.pdf. – Загол. з екрана.
9. MAX7219/MAX7221 Serially Interfaced, 8-Digit LED Display Drivers [Електронний ресурс]. – Режим доступу : https://datasheets. maximintegrated.com/en/ds/MAX7219-MAX7221.pdf. – Загол. з екрана.
10. ULN2002A/ ULN2003A/ ULN2004A High Voltage, High Current Darlington Transistor Arrays [Електронний ресурс]. – Режим доступу : https://www.diodes.com/assets/Datasheets/ULN200xA.pdf. – Загол. з екрана.
11. Unipolar Stepper Motors vs Bipolar Stepper Motors [Електронний ресурс]. – Режим доступу : https://www.circuitspecialists.com/blog/ unipolar-stepper-motor-vs-bipolar-stepper-motors/. – Загол. з екрана.
12. How Do Servo Motors Work? [Електронний ресурс]. – Режим доступу : https://www.jameco.com/jameco/workshop/howitworks/ how-servo-motors-work.html. – Загол. з екрана.

ДЛЯ НОТАТОК

Навчальне видання

Ярослав Михайлович
КРАЙНИК

Комп'ютерні системи

Методичні вказівки до виконання лабораторних робіт

Випуск 258

Редактор *Я. Котенко*.
Технічний редактор, комп'ютерна верстка *Д. Кардаш.*
Друк *С. Волинець.* Фальцювально-палітурні роботи *О. Кутова.*

Підп. до друку 21.11.2018
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура "Times New Roman". Друк ризограф.
Ум. друк. арк. 2,79. Обл.-вид. арк. 1,11.
Тираж 5 пр. Зам. № 5599.

Видавець і виготовлювач: ЧНУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.