МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

В.О.Денисюк, С. М. Цирульник

МІКРОПРОЦЕСОРНІ СИСТЕМИ УПРАВЛІННЯ

Навчальний посібник

Вінниця - 2021

УДК 004.272.43(075)
ББК 32.971.32-043я73
М59

Рекомендовано Вченою радою як навчальний посібник для студентів галузі знань 12 «Інформаційні технології» (протокол № 14 від 14.04.2021р.)

Рецензенти:
А.О.Семенов – доктор технічних наук, професор кафедри радіотехніки ВНТУ;
О.М.Джеджула – доктор педагогічних наук, професор, завідувач кафедри математики, фізики та комп'ютерних технологій ВНАУ;
В.В.Мартинюк – доктор технічних наук, професор, завідувач кафедри автоматизацій, комп'ютерно-інтегрованих технологій і телекомунікацій ХНУ.

М59 Денисюк В.О., Цирульник С.М. Мікропроцесорні системи управління: навч. посіб../ В.О.Денисюк, С.М.Цирульник; Вінн. нац. аграр. ун-т. - Вінниця:ВНАУ, 2021. - 200 с.

ISBN

Зміст видання відповідає освітньому рівню бакалавр галузі знань 12 Інформаційні технології і програмі дисципліни «Мікропроцесорні системи управління».

Розглянуті практичні аспекти побудови та проектування мікропроцесорних систем управління на апаратно-програмній платформі Arduino. Запропонований комплексний підхід, який поєднує комп'ютерне моделювання роботи віртуального стенда та реального стенду Arduino Learner Kit, що дозволяє підвищити ефективність лабораторно-практичних занять та зменшити матеріальні витрати на придбання, обслуговування та ремонт лабораторного обладнання.

УДК 004.272.43(075)

ISBN

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ УПРАВЛІННЯ 8
1.1. Основні поняття мікропроцесорних систем 8
1.2. Архітектура AVR мікроконтролерів 15
1.3. Система команд і програмна модель AVR 19
1.4. Програмування в машинних кодах 25
1.5. Порти введення/виведення AVR. Програмне введення/виведення інформації 30
1.6. Таймери/лічильники. Модуль переривань 32
1.7. Мікроконтролери Arduino та ESP8266 35
1.8. Аналого-цифрові перетворювачі. Цифро-аналогові перетворювачі 37
РОЗДІЛ 2. ФУНКЦІОНАЛЬНІ ВУЗЛИ МІКРОПРОЦЕСОРНИХ СИСТЕМ 42
2.1. Особливості живлення та формування тактової частоти 42
2.2. Виконавчі пристрої мікропроцесорних систем управління 48
2.3. Елементи індикації 52
2.4. Кнопки та датчики. Оптичні датчики 57
2.5. Пристрої формування звукових сигналів. Пристрої управління двигунами постійного струму 61
2.6. Периферійний послідовний інтерфейс UART, SPI 67
2.7. Організація обміну даними інтерфейсом I^2C, 1-Wire 69
2.8. Організація обміну між ПК та МК по інтерфейсу USB 75
РОЗДІЛ 3. ЛАБОРАТОРНИЙ ПРАКТИКУМ 79
3.1. Практична робота №1. Вивчення роботи портів вводу-виводу плати Arduino 89
3.2. Практична робота №2. Вивчення роботи переривань, ШІМ та АЦП програмованого мікроконтролера Arduino 94
3.3. Практична робота №3. Робота з RGB світлодіодом 97

3.4. Практична робота №4. Робота з семисегментним індикатором та АЦП програмованого мікроконтролера Arduino 101
3.5. Практична робота №5. Семисегментний індикатор з регістром зсуву 74HC595 107
3.6. Практична робота №6. Робота з LCD – дисплеєм 110
3.7. Практична робота №7. Програмування Arduino. Дослідження роботи датчика температури LM35 115
3.8. Праткична робота №8. Програмування Arduino. Дослідження роботи датчика температури та вологості DHT11 119
3.9. Практична робота №9. Робота з інтерфейсом TWI (I2C) та годинником реального часу DS1307 123
3.10. Практична робота №10. Робота з матричним світлодіодним індикатором 129
ЛІТЕРАТУРА 136
Додаток А. Лабораторний макет «Arduino Learner Kit». Схема електрична принципова 138
Додаток Б. Програма LED1, LED2 141
Додаток В. Програма Tone, LED3 − LED5 142
Додаток Г. Програма RGB1 − RGB5 145
Додаток Д. Програма SEG1 − SEG6 150
Додаток Е. Програма SHIFT 164
Додаток Ж. Програма LCD1 − LCD7 167
Додаток И. Програма LM35_SEG, LM35_Shift, LM35_LCD 174
Додаток К. Програма DHT11_SEG, DHT11_LCD 184
Додаток Л. Програма DS1307_LCD, DS1307_SEG 190
Додаток М. Програма Matrix_One, Matrix_Smile 198

3.1 Практична робота №1. Вивчення роботи портів вводу-виводу плати Arduino

Мета: навчитися програмувати Arduino та дослідити роботу портів вводу-виводу мікроконтролера.

Завдання: створити програму керування світлодіодами, використовуючи порти вводу-виводу Arduino.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з'єднувачі.

Теоретичні відомості

Мікроконтролер — це логічне пристрій, який створено для управління іншими пристроями за допомогою логічних (цифрових) сигналів. Це означає, що максимум можна зняти з порта 40 мА, а рекомендується не більше 20 мА. Що станеться, якщо зняти з порта більше, ніж він може віддати? Він зламається. Що буде, якщо зняти з декількох портів більше, ніж може віддати мікроконтролер в цілому? Згорить мікроконтролер. Тому нічого потужніше світлодіода та маленькою пищалки до мікроконтролера підключати не можна. Ніяких моторчиків, лампочок, нагрівачів, потужних радіо-модулів та іншого живити від цифрових портів не можна. Цифрові порти служать для подачі команд іншим пристроям, наприклад реле / транзисторів для комутації навантажень.

Цифровий порт може перебувати в двох станах, вхід та вихід. Режим роботи вибирається за допомогою функції pinMode (pin, mode), де pin це номер порта, а mode це режим роботи (INPUT — вхід, OUTPUT — вихід, INPUT_PULLUP —вхід підтягнутий до живлення (PushUp)).

За замовчуванням всі порти налаштовані як входи (INPUT). Для зміни режимів роботи портів D2-D13, A0 -A5 можна використати фрагмент програми:

```
for ( byte i = 2; i < = 19; i ++ ) {
       pinMode (i, OUTPUT ) ; // робимо виходами }
```

Для формування цифрового сигналу використовується функція digitalWrite (pin, value), де pin це цифровий порт Arduino, підписаний на платі як

D; value — рівень сигналу: HIGH високий, LOW низький. Також можна використовувати цифри 0 та 1. Приклад, в якому порти ініціалізуються як виходи, і на них подається сигнал:

```
void setup () {
  pinMode (10, OUTPUT )   ; // D10 як вихід
  pinMode (A3, OUTPUT )   ; // A3 як вихід
  pinMode (19, OUTPUT )   ; // A5 як вихід (Nano / UNO)
  digitalWrite (10, HIGH) ; // високий рівень на D10
  digitalWrite (A3, 1)    ; // високий рівень на A3
  digitalWrite (19, 1)    ; // високий рівень на A5
}
void loop () {}
```

Порт, налаштований як OUTPUT, за замовчуванням має сигнал LOW. Цифровий порт може «вимірювати» напругу, але повідомити він може тільки про її відсутність (сигнал низького рівня, LOW) або наявність (сигнал високого рівня, HIGH). Відсутність напруги вважається проміжок від 0 до 2,1V. Відповідно від 2.1V до VCC (до 5V) мікроконтролер вважає за наявність сигналу високого рівня. Таким чином мікроконтролер може працювати з логічними пристроями, які шлють йому високий сигнал з напругою 3.3V, він такий сигнал прийме як HIGH. Не можна подавати на цифровий порт напругу вище напруги живлення мікроконтролера. Для читання рівня сигналу на порту використовується функція digitalRead (pin), де pin це номер порта на платі Arduino. Вони підписані як D , а також порти A0-A5. Функція повертає 0, якщо сигнал низького рівня, і 1 якщо сигнал високого рівня. Приклад:

```
void setup () {
  Serial. begin ( 9600 ) ;
}
void loop () {
  Serial. println ( digitalRead ( 5 )) ;
}
```

Регістри портів дозволяють низькорівневі високошвидкісні маніпуляції з портами мікроконтролера. Мікроконтролери, що використовуються в Arduino мають три порти : B (D8-D13), C(A0-A7), D(D0-D7) (рис. 1.10).

Кожен порт контролюється трьома регістрами, кожен з яких відповідає за певний стан. Регістр DDR визначає, який біт порта вхідний, а який вихідний. Регістр PORT встановлює біт порта у відповідний стан HIGH або LOW, регістр PIN читає стан вхідного порта.

Регістри DDR та PORT можуть бути як прочитані, так і записані. Регістр PIN відповідає за стан вхідних портів, тому може бути лише прочитаний.

PORTD відповідає за виводи 0 - 7.

DDRD — регістр напряму порту D;

PORTD — регістр даних порту D;

PIND — регістр вхідних даних порту D.

PORTB відповідає за виводи 8 - 13. Два старших біта (6 та 7), що відповідають за виводи кварцу, не використовуються.

DDRB — регістр напряму порту B;

PORTB — регістр даних порту B;

PINB — регістр вхідних даних порту B.

PORTC відповідає за аналогові виводи 0 - 5.

DDRC — регістр напряму порту C;

PORTC — регістр даних порту C;

PINC — регістр вхідних даних порту C.

Кожен біт в цих регістрах відповідає за відповідний вивід, так молодший біт у DDRB, PORTB, та PINB посилається на вивід PB0 (цифровий порт D8) (рис. 1.4). Слід пам'ятати, що виводи D0 та D1 задіяні послідовним портом та робота з ними можлива тільки в тому випадку, якщо налагодження та послідовний порт не потрібні. Приклад роботи з портом D:

```
// призначаємо виводи Arduino 1-7 вихідними, вивід 0-вхідним
DDRD = B11111110;
// виводи з 2 по 7 вихідні, стан виводів 0 та 1 не змінюється
DDRD = DDRD | B11111100;
// встановлюємо рівень HIGH на цифрових виводах 7,5,3
PORTD = B10101000;
```

Приклад роботи з портом B:

```
void setup() {
//виставляємо всі біти порту B як вихід, PB4 як вхід
DDRB = B11101111;
PORTB = B00000000; //скидаємо всі біти порту B
}
void loop() {
if (PINB==B00010000) {
PORTB |= 1 << 5 // PB5=1
delay (1000) // очікуємо секунду
PORTB &= ~(1 << 5)// PB5=0
delay (1000) }
}
```

При роботі з портами можна використовувати вбудовані в Arduino функції для роботи з бітами порта bitRead(), bitWrite(), bitSet(), bitClear().

bitRead(x, n) зчитує стан зазначеного біта числа, де *x*: регістр вхідних даних порту (PINB, PIND, PINC) біт якого буде зчитуватись; *n*: номер біта, стан якого необхідно зчитати (стан біту (0 або 1)).

bitWrite(x, n, b) змінює стан зазначеного біта змінної, де *x*: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); *n:* номер біта, стан якого необхідно змінити; *b:* нове значення біта (0 або 1).

bitSet(x, n) встановлює зазначений біт (записує 1) числової змінної, де *x*: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); *n:* номер біта, стан якого необхідно змінити.

bitClear(x, n) скидає вказаний біт (записує 0) числової змінної, де *x*: числова змінна, у якій необхідно змінити біт (DDRx або PORTx); *n:* номер біта, стан якого необхідно змінити.

Використання регістрів портів суттєво зменшує розмір коду та збільшує швидкодію на декілька порядків.

Хід виконання роботи

1. Виконати з’єднання світлодіодів з цифровими виводами Arduino (HL1-HL6 →D2-D7).
2. Підключити схему до комп’ютера через USB порт плати Arduino.
3. Завантажити програму LED1 (додаток Б) до лабораторного макету /

віртуального стенду та дослідити роботу програми.

4. Завантажити програму LED2 (додаток Б) до лабораторного макету / віртуального стенду та дослідити роботу програми.

Завдання

1. Реалізувати на світлодіодах HL1-HL6 ефект вогню, що «біжить».
2. Реалізувати на світлодіодах HL1-HL6 ефект тіні, що «біжить».
3. Реалізувати на світлодіодах HL1-HL6 ефект вогню/тіні, що «біжить», із зміною напрямку на протилежний.
4. Варіанти підключення світлодіодів до виходів лабораторного макету взяти з таблиці 3.1. Написати програму включення-виключення світлодіодів з заданою частотою у порядку згідно варіанту (двома способами, як в програмі LED1, LED2). Налагодити програму в середовищі Arduino і перевірити на лабораторному макеті/ віртуальному стенді.

Таблиця 3.1 – Варіанти індивідуальних завдань

№ варіанту	Номера виводів плати Arduino				Початкова частота миготіння, Гц	Порядок включення світлодіодів
	HL1	HL2	HL3	HL4		
1	11	6	10	9	0,5	одночасно
2	10	5	3	6	1	збільшення
3	9	3	6	11	2	зменшення
4	11	3	9	10	0,5	через один
5	10	6	8	11	1	одночасно
6	9	5	7	10	2	збільшення
7	9	10	6	11	0,5	зменшення

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, скріншоти, висновки, перелік посилань).

Контрольні питання

1. Зазначте основні сфери застосування Arduino?

2. Які є основні функції цифрового введення та виведення?
3. Які значення має параметр value функції *digitalWrite ()*?
4. Чи повертає значення функція *pinMode (pin, mode)*?
5. Описати алгоритм ініціалізації порту вводу-виводу Arduino?
6. Чому в лабораторній роботі світлодіоди підключені без використання транзисторів?
7. Що виконує інструкція ++pin?
8. У чому різниця між змінними int та unsigned int?
9. Що повертає функція millis()?

3.2 Практична робота №2. Вивчення роботи переривань, ШІМ та АЦП програмованого мікроконтролера Arduino

Мета: дослідити можливості створення мелодії, за допомогою спеціальних бібліотек, в Arduino; дослідити роботу переривань, АЦП та ШІМ в контролерах Arduino.

Завдання: написати програму створення мелодії, створити програми керування світлодіодами та регулювання їх яскравості з використанням тактових кнопок, зовнішніх переривань, АЦП та ШІМ лабораторного макету / віртуального стенду «Arduino Learner Kit» Arduino.

Обладнання: лабораторний макет / віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

Для відтворення звукового сигналу, використовуючи мікроконтролер Arduino, використовують команду *tone()*, яка генерує на виводі мікроконтролера прямокутний сигнал заданої частоти (з коефіцієнтом заповнення 50%). Функція також дозволяє задавати тривалість сигналу. Однак, якщо тривалість сигналу не вказана, він буде генеруватися доти, поки не буде викликана функція *noTone()*. Для відтворення звуку порт Arduino можна підключити до зумеру або динаміку.

У кожен момент часу може генеруватися тільки один сигнал заданої

частоти. Якщо сигнал вже генерується на будь-якому виводі, то використання функції *tone()* для цього виводу призведе до зміни частоти цього сигналу. У той же час виклик функції *tone()* для іншого виводу не матиме ніякого ефекту. Для відтворення різних звуків на кількох виводах, необхідно спершу викликати *noTone()* на одному і тільки після цього використовувати функцію *tone()* на наступному виводі.

tone (pin, frequency) / tone (pin, frequency, duration).Параметри:

- pin: вивід, на якому буде генеруватися сигнал;
- frequency: частота сигналу в Герцах (unsigned int);
- duration: тривалість сигналу в мілісекундах (unsigned long);
- значення, що повертає – немає.

Приклад формування тонального сигналу наведений у додатку В (програма Tone).

Переривання – це сигнали, що переривають нормальний перебіг програми. Вони використовуються для апаратних пристроїв, що вимагають негайної реакції на появу подій. Обробка переривань у мікроконтролері відбувається за допомогою модуля переривань, який приймає запити переривання й організовує перехід до виконання визначеної програми. Запити переривання можуть надходити як від зовнішніх джерел, так і від джерел, розташованих у різних внутрішніх модулях мікроконтролера. Тактові кнопки досить часто підключають до входів зовнішніх переривань (0 – pin D2, 1 – pin D3). Для роботи з зовнішніми перериваннями використовують функції Arduino *attachInterrupt (interrupt, function, mode), detachInterrupt (pin), interrupts ()*. Більш детально ці функції описані в розділі 1.6 [13, 14].

Для роботи з АЦП та ШІМ використовуються аналогові функції портів A0-A7 Arduino Nano: *analogReference (type), analogRead (pin), analogWrite (pin, value).* Більш детально ці функції описані в розділі 2.5 [13, 14].

У додатку В наводиться програма LED3, яка демонструє роботу з зовнішнім перериванням та LED4, яка використовує формування ШІМ сигналу для зміни яскравості світіння світлодіода. У програмі LED5 зчитується значення

аналогової напруги з RV1. Це значення використовується для формування частоти мигання світлодіода та зміни його яскравості.

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму Tone (додаток В) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання елементу «buzzer» та тактових кнопок S1-S4 у відповідність до програми. Дослідити роботу програми.

3. Завантажити програму LED3 (додаток В) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання HL1-HL6 та тактових кнопок S1-S4 у відповідність до програми. Дослідити роботу програми.

4. Завантажити програму LED4 (додаток В) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання HL1-HL6 та тактових кнопок S1-S4 у відповідність до програми. Дослідити роботу програми.

5. Завантажити програму LED5 (додаток В) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання HL1-HL6 та тактових кнопок S1-S4 у відповідність до програми. Дослідити роботу програми.

Завдання

1. Реалізувати програму, у якій кожній кнопці призначена своя нота або мелодія.

2. Реалізувати програму з застосуванням переривань INT0, INT1. При натисканні кнопки, що підключена до INT0 збільшується тональність звукового сигналу, а при натисканні кнопки, що підключена до INT1 зменшується тональність звукового сигналу.

3. Реалізувати програму, у якій тональність звукового сигналу встановлюється потенціометром RV1.

4. Реалізувати на світлодіодах HL1-HL6 програму, у якій кожний з них світиться з різною яскравістю (використовуються порти D3, D5, D6, D9, D10,

D11).

5. Реалізувати на світлодіодах HL1-HL6 програму (використовуються порти D3, D5, D6, D9, D10, D11), у якій кнопками S1/S4 збільшується/зменшується яскравість їх світіння.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Що таке переривання? Які типи переривань використовуються в Arduino?
2. Які параметри функції *attachInterrupt ( )*?
3. Для чого використовується директива *volatile*?
4. Які параметри функції *analogReference ( )?*
5. Які основні команди для роботи з АЦП та ШІМ?
6. Яку команду використовує Arduino для відтворення звуку? Опишіть основні її особливості.
7. Які параметри має функція *tone()*?

3.3 Практична робота №3. Робота з RGB світлодіодом

Мета: дослідити роботу RGB світлодіода, закріпити навички роботи з цифровими портами, тактовими кнопками, формуванням ШІМ; навчитися програмувати режими роботи Arduino з використанням масивів.

Завдання: створити програму керування RGB світлодіодом з використанням тактових кнопок, зовнішніх переривань, АЦП та ШІМ лабораторного макету / віртуального стенду «Arduino Learner Kit» Arduino.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

Масив — це набір змінних, доступ до яких здійснюється за індексом. Для роботи з масивом його потрібно створити (оголосити). Способи створення

(оголошення) масивів:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

Можна оголосити масив без його ініціалізації, як *myInts*. У *myPins* оголошений масив без прямої вказівки його розміру. Компілятор сам порахує елементи та створить масив відповідного розміру. При створенні (ініціалізації) масиву можна вказати його розмір, як *mySensVals*. При оголошенні масиву типу *char*, в ньому необхідно місце для зберігання обов'язкового нульового символу, тому розмір масиву повинен бути на один символ більше.

Нумерація елементів масиву починається з нуля , тобто перший елемент масиву має індекс 0. Так *mySensVals[0] ==2*, *mySensVals[1] ==4*.

Це також означає, що в масиві з 10 елементів, останній елемент має індекс 9. Отже:

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
// myArray[9] дорівнює 11
// myArray[10]  буде помилка
```

Тому, необхідно бути уважним при зверненні до масивів. Звернення до елементу за межами масиву (коли зазначений індекс більше, ніж оголошений розмір масиву – 1) призведе до читання даних з комірки пам'яті, що використовується для інших цілей. Зчитування з цієї області призведе до збою в роботі програми. На відміну від BASIC або JAVA, компілятор C не перевіряє правильність індексів при зверненні до елементів масиву.

Запис значення до масиву — *mySensVals[0] = 10*. Зчитування запису з масиву — *x = mySensVals[4]*.

Робота з масивами часто здійснюється всередині циклів FOR, в яких лічильник циклу використовується як індексу кожного елемента масиву. Наприклад, програма налаштування режимів роботи портів Arduino може виглядати так:

```
const byte rgbPins[3] = {11,10,9};
```

```
void setup() {
    for( byte i=0; i < 3; i++ )
        pinMode( rgbPins[i], OUTPUT );
}
```

В одномірних масивах елементи визначаються просто порядковим номером. У двовимірних масивах (матриця або таблиця) кожен елемент має номер рядка та стовпця. Задається такий масив ось так:

```
// двовимірний масив, 5 рядків 10 стовпців
byte myTable [5][10] ;
// матриця 3x3
byte myMatrix [3][3] = {
  { 10, 11, 12 } ,
  { 13, 14, 15 } ,
  { 16, 17, 18 } ,
} ;
```

Після останнього члена масиву можна ставити кому, це не призведе до помилки (приклад коду вище). У розглянутому вище двовимірному масиві елемент з адресою 0, 2 (рядок 0 стовпець 2) *myMatrix [0] [2]* має значення 12.

Дребезг контактів — це явище, що відбувається в електромеханічних пристроях (кнопках, реле, герконах, перемикачах, контакторах), що триває деякий час після замикання електричних контактів. Після замикання (натискання кнопки, включення реле і т.д.) відбуваються багаторазові неконтрольовані замикання та розмикання контактів за рахунок пружності матеріалів та деталей контактної системи. Перехідні процеси протікають дуже швидко (від 0,5 до декількох сотень мілісекунд). Тому їх не помічаємо, наприклад, коли включаємо світло в кімнаті. Лампа розжарювання не може змінювати свою яскравість з такою швидкістю. Але, обробляючи сигнал від кнопки на швидкому пристрої, як Arduino, повинні враховувати при програмуванні це явище.

Найпростішим способом боротьби з дребезгом кнопки є витримування паузи. Для цього робимо паузу 10-50 мілісекунд, як наведено у прикладі програми нижче.

```
int currentValue, prevValue;
void loop ( ) {
  currentValue = digitalRead (PIN_BUTTON) ;
```

```
if ( currentValue! = prevValue ) {
  delay ( 10 ) ;
  currentValue = digitalRead (PIN_BUTTON) ;
}
prevValue = currentValue;
Serial. println (currentValue) ; }
```

Проблема з дребезгом настільки актуальна, що є спеціальні бібліотеки, в яких не потрібно організовувати очікування та паузи вручну — це все робиться всередині спеціального класу. Приклад популярної бібліотеки для боротьби з дребезгом кнопок — бібліотека Bounce.

Більш правильним способом боротьби з дребезгом є використання апаратного рішення, що згладжує імпульси з кнопки. Апаратний спосіб усунення дребезгу заснований на використанні RC фільтру або тригера Шмідта [9, 10].

RGB світлодіод на схемі лабораторного макету позначений HL7 (рис. 3.1) та підключений за схемою з загальним катодом. Для його з'єднання з Arduino Nano використовується з'єднувач X6 (рис. 3.7).

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.
2. Завантажити програму RGB1 (додаток Г) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання RGB світлодіода у відповідність до програми. Дослідити роботу програми.
3. Завантажити програму RGB2 (додаток Г) до лабораторного макета / віртуального стенду. Дослідити роботу програми.
4. Завантажити програму RGB3 (додаток Г) до лабораторного макета / віртуального стенду. Дослідити роботу програми.
5. Завантажити програму RGB4 (додаток Г) до лабораторного макета / віртуального стенду. Дослідити роботу програми.
6. Завантажити програму RGB5 (додаток Г) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання RGB світлодіода та

тактової кнопки у відповідність до програми. Дослідити роботу програми.

Завдання

1. Внести зміни до програми RGB5 (додаток Г). Для опрацювання моментів натиснення кнопки використати зовнішнє переривання. Для вибору режиму роботи RGB світлодіода використати оператор SWITCH.

2. Реалізувати програму з застосуванням переривань INT0, INT1. При натисканні кнопки, що підключена до INT0 змінюється колір світлодіода, а при натисканні кнопки, що підключена до INT1 змінюється частота мигання світлодіода даним кольором.

3. Реалізувати програму, у якій колір RGB світлодіода змінюється потенціометром RV1.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань).

Контрольні питання

1. Що таке дребезг контактів? Як програмно усунути його?
2. Як апаратно усунути дребезг контактів? Наведіть практичні схеми.
3. Для чого використовується бібліотека Bounce? Наведіть приклад її застосування
4. Реалізуйте програму «вогонь, що біжить» з використанням одномірного масиву.
5. Реалізуйте програму «вогонь, що біжить» з використанням двомірного масиву.

3.4 Практична робота №4. Робота з семисегментним індикатором

Мета: ознайомитись з принципом роботи семисегментного індикатора та дослідити можливості програмування його роботи у динамічному режимі; навчитися програмувати режими роботи семисегментного індикатора з

використанням переривань таймера/лічильника, що входить до складу мікроконтролера Arduino; закріпити навички роботи з цифровими портами, тактовими кнопками, масивами.

Завдання: створити програму відображення інформації на семисегментному індикаторі з використанням тактових кнопок, внутрішніх переривань, АЦП лабораторного макету / віртуального стенду «Arduino Learner Kit» Arduino.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з'єднувачі.

Теоретичні відомості

Плата Arduino дозволяє швидко та мінімальними засобами вирішити найрізноманітніші завдання. Але там де потрібні довільні інтервали часу (періодичне опитування датчиків, високоточні ШІМ сигнали, імпульси великої тривалості) стандартні бібліотечні функції затримки не зручні. На час їх дії скетч призупиняється і керувати ним стає неможливо. У подібній ситуації краще використовувати вбудовані AVR таймери. Таймери, як і зовнішні переривання, працюють незалежно від основної програми.

У стандартних платах Arduino є три таймера Timer0, Timer1 і Timer2. Timer0 є 8 бітним таймером, це означає, що його рахунковий регістр може зберігати числа до 255. Timer0 використовується стандартними часовими функціями Arduino такими як delay () і millis (), так що краще його не використовувати у своїх проектах.

Timer1 це 16 бітний таймер з максимальним значенням 65535. Цей таймер використовує бібліотека Arduino Servo, враховуйте це якщо застосовуєте його в своїх проектах.

Timer2 — 8 бітний і дуже схожий на Timer0. Він використовується в функції tone () Arduino.

Для обробки переривань у мові програмування Arduino використовується функція ***ISR ()***. У ній необхідно вказати тип переривання [1, 10]. Arduino

(ATmega328P) використовує такі варіанти параметра функції *ISR ()* для роботи з таймерами:

- IMER2_COMPA_vect — переривання від Timer2 при збігу з A;
- TIMER2_COMPB_vect — переривання від Timer2 при збігу з B;
- TIMER2_OVF_vect — переривання переповнення Timer2;
- TIMER1_CAPT_vect — переривання від Timer1 (режим захоплення);
- TIMER1_COMPA_vect — переривання від Timer1 при збігу з A;
- TIMER1_COMPB_vect — переривання від Timer1 при збігу з B;
- TIMER1_OVF_vect — переривання переповнення Timer1;
- TIMER0_COMPA_vect — переривання від Timer0 при збігу з A;
- TIMER0_COMPB_vect — переривання від Timer0 при збігу з B;
- TIMER0_OVF_vect — переривання переповнення Timer0.

Для того щоб використовувати таймери в AVR є регістри налаштувань. Таймери містять безліч таких регістрів. Два з них — регістри управління таймера / лічильника містять установчі змінні й називаються TCCRxA і TCCRxB, де x — номер таймера (TCCR1A і TCCR1B). Кожен регістр містить 8 біт і кожен біт зберігає конфігураційну змінну. Найбільш важливими є три останні біта в TCCR1B: CS12, CS11 і CS10. Вони визначають тактову частоту таймера (табл. 3.2). За замовчуванням ці біти не встановлені.

Таблиця 3.2 — Біти конфігурації частоти роботи таймера/лічильника

CS12	CS11	CS10	Опис
0	0	0	Таймер лічильник 1 зупинений
0	0	1	СК
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Зовнішній вхід T1, спадаючий
1	1	1	Зовнішній вхід T1, наростаючий

TIMSK1 це регістр маски переривань Timer1. Він контролює

переривання, які таймер може викликати. Установка біта 0 біту (TOIE1) вказує таймеру, що дозволено переривання коли таймер переповнюється (дораховує до максимального значення з частотою, що визначена бітами CS12, CS11, CS10). Якщо частота предільника Timer1 (табл. 3.2) встановлена у значення 001, то при тактовій частоті 16 МГц Atmega328 переривання виникне приблизно через 0,0041 секунд (65535/16МГц).

Приклад програми з використанням переривання від Timer1 у режимі переповнення:

```
#define LEDPIN 13
int  count = 100;
void setup(){
  pinMode(LEDPIN, OUTPUT);
  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
  TCNT1  = 0;  // set Timer1 preload value to 0
  TIMSK1 = 1;  // enable Timer1 overflow interrupt

  ISR(TIMER1_OVF_vect) {
     digitalWrite(LEDPIN, !digitalRead(LEDPIN));
  }
 }
void loop(){
  if(digitalRead(button) == 0){
    count++;  // increment 'count' by 1
    if(count == 9999){
      count = 0;}
    delay(200);  // wait 200 milliseconds
  }
}
```

TCNT1 це регістр, який рахує імпульси. У програмі йому присвоєне значення 0, а це значить, що переривання виникне, коли він дорахує до значення 65535. Запуск лічби починається встановленням біту CS10 (TCCR1B = 1) і, як тільки виникає переривання у режимі переповнення, викликається ISR (TIMER1_OVF_vect). Це відбувається завжди коли таймер переповнюється.

Функція **random(max) / random(min, max)** генерує псевдо-випадкові

числа від min до max–1 (від 0 до 4 294 967 295).

```
long randNumber;

void setup(){
  Serial.begin(9600);
}

void loop() {
  // виводимо випадкове число у діапазоні від 0 до 299
  randNumber = random(300);
  Serial.println(randNumber);

  // виводимо випадкове число у діапазоні від 10 до 19
  randNumber = random(10, 20);
  Serial.println(randNumber);
  delay(50);
}
```

Схема 4 розрядного семисегментного LED-індикатора наведена на рис. 3.2. Чотирьох розрядний семисегментний індикатор має 12 контактів (з'єднувач X2). 8 контактів призначені для 8 світлодіодів на кожному з семисегментних індикаторів; інші 4 контакти призначені для вибору розряду індикатора. Індикатор виконаний за схемою з загальним катодом. Для запалювання сегменту індикатора подається рівень HIGH. Транзистори T1-T4 (BC547) використовуються як комутатори. Транзистор увімкнений, коли на базу подається позитивна напруга (рівень HIGH). Для обмеження струму транзистора використовуються резистори R9-R12 (4,7 кОм). Індикатор працює в динамічному режимі та використовує інерційність зору людини. Частота перемикання розрядів індикатора не менше 100 Гц (25 Гц на кожний розряд).

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму SEG1 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного

індикатора у відповідність до програми. Дослідити роботу програми.

3. Завантажити програму SEG2 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного індикатора у відповідність до програми. Дослідити роботу програми..

4. Завантажити програму SEG3 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного індикатора у відповідність до програми. Дослідити роботу програми..

5. Завантажити програму SEG4 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного індикатора у відповідність до програми. Дослідити роботу програми..

6. Завантажити програму SEG5 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного індикатора у відповідність до програми. Дослідити роботу програми.

7. Завантажити програму SEG6 (додаток Д) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання семисегментного індикатора та тактової кнопки у відповідність до програми. Дослідити роботу програми.

Завдання

1. Внести зміни до програми SEG5 (додаток Д). Вивести двійковий код напруги, що зчитується АЦП Arduino з потенціометру RV1.

2. Внести зміни до програми SEG6 (додаток Д), щоб за натисненням кнопки кожен раз відображалось випадкове число.

3. Реалізувати програму з застосуванням тактових кнопок S1-S4. Кожна кнопка встановлює число від 0 до 9 у відповідній позиції індикатора.

4. Реалізувати програму з застосуванням тактових кнопок S1-S4. Кнопки S3, S4 реалізують інкремент / декремент у старшій позиції індикатора в діапазоні від 0 до 99. Кнопки S1, S2 виконують аналогічну дію над числом у молодшій позиції індикатора.

5. Реалізувати програму з застосуванням тактових кнопок S1-S4. Кнопки

S3, S4 реалізують інкремент / декремент у старшій позиції індикатора в діапазоні від 0 до 23. Кнопки S1, S2 виконують аналогічну дію над числом в діапазоні від 0 до 59 у молодшій позиції індикатора.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Яке призначення функції randomSeed(analogRead(0))?
2. Які існують апаратно-програмні способи генерування випадкової послідовності?
3. Яка має бути частота перемикання розрядів індикатора? Яка вона генерується в програмі SEG1 та SEG6?
4. Які існують режими роботи Timer1? Як організувати переривання в режимі порівняння?
5. Як налаштувати переривання кожні 100мс від Timer 1?
6. Які зміни потрібно внести до програми SEG2, якщо індикатор буде з загальним анодом?

3.5 Практична робота №5. Семисегментний індикатор з регістром зсуву 74HC595

Мета: ознайомитись з принципом роботи регистру зсуву 74HC595 разом з семисегментим індикатором та дослідити можливості програмної реалізації SPI інтерфейсу; закріпити навички програмування режимів роботи семисегментного індикатора з використанням переривань таймера/лічильника, що входить до складу мікроконтролера Arduino; закріпити навички роботи з цифровими портами, тактовими кнопками, масивами.

Завдання: створити програму відображення інформації на семисегментному індикаторі з використанням регистру зсуву 74HC595, тактових кнопок, внутрішніх переривань, АЦП лабораторного макету / віртуального

стенду «Arduino Learner Kit» Arduino.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з'єднувачі.

Теоретичні відомості

74HC595 — це регістр зсуву послідовно-паралельно типу (SIPO), який використовується для збільшення кількості виходів з мікроконтролера. Схема модуля регістра зсуву 74HC595 наведена на рис. 3.3.

Мікросхема 74HC595 перетворює послідовний сигнал, що входить, на 1 лінію (SER) у вихідний паралельний сигнал на 8 виводах (Qx). Послідовна передача синхронна: для тактових сигналів використовується вивід (SCK). Також окремим виводом (RCK) управляється регістр даних, що дозволяє змінювати сигнал на 8 виходах одночасно, коли усі дані передані.

Таким чином, 3 портами мікроконтролера можна керувати 8 цифровими виходами. З регістрів 74HC595 можна робити каскади, підключаючи один до одного (через пін QH*). Це дозволяє отримати 16, 24, 32 цифрові виходи.

Для зручної роботи з регістром 74HC595 в Arduino існує вбудована функція **shiftOut()**. Вона здійснює побітовий зсув та вивід байту даних, починаючи з найстаршого (лівого) або молодшого (правого) значущого біта. Функція по черзі відправляє кожен біт на вказаний вивід даних, після чого формує імпульс (високий рівень, потім низький) на тактовому виводі, повідомляючи зовнішньому пристрою про надходження нового біта.

Функція є програмною реалізацією SPI.

shiftOut(dataPin, bitOrder, value). Параметри:

- dataPin: вхід данних у послідовному коді (int);
- clockPin: тактовий вивід (int);
- bitOrder: характеризує порядок, в якому будуть зсуватися та виводитися біти; може приймати значення MSBFIRST (старший біт перший) або LSBFIRST (молодший біт перший);
- value: байт даних (byte).

Приклад передачи данних з використанням функції shiftOut():

```
#define clock 13
#define data 12
#define latch 10

void setup() {
  pinMode(clock, OUTPUT);
  pinMode(data, OUTPUT);
  pinMode(latch, OUTPUT);
  digitalWrite(latch, HIGH);
}
void loop() {
  digitalWrite(latch, LOW);
  shiftOut(data, clock, LSBFIRST, 0b10000000);
  digitalWrite(latch, HIGH);
}
```

Для роботи з 74HC595 виводи 16 (VCC) та 10 (SRCLR) повинні бути підключені до 5 В, а виводи 8 (GND) та 13 (OE) повинні бути підключені до землі. Для цього необхідно перевести перемикач 2 SW1 в положення «On». Контакти 11, 12, 14 необхідно з'єднати з трьома цифровими портами Arduino через з’єднувач X4. З’єднувач X3 потрібно з’єднати з відповідними контактами (1-8) X2 (рис. 3.2).

Хід виконання роботи

1. Підключити схему до комп’ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму SHIFT (додаток Е) до лабораторного макета / віртуального стенду, попередньо виконати з’єднання Arduino, регістра зсуву 74HC595, семисегментного індикатора та тактової кнопки у відповідності до програми. Дослідити роботу програми.

Завдання

1. Внести зміни до програми SHIFT (додаток Е). Вивести двійковий код напруги, що зчитується АЦП Arduino з потенціометру RV1.

2. Внести зміни до програми SHIFT (додаток Е), щоб за натисненням кнопки кожен раз відображалось випадкове число.

3. Реалізувати програму з застосуванням тактових кнопок S1-S4 та регістру зсуву. Кожна кнопка встановлює число від 0 до 9 у відповідній позиції індикатора.

4. Реалізувати програму з застосуванням регістру зсуву 74HC595, яка відображає на семисегментному індикаторі напис «LOAD». За натисненням кнопки S1 відбражається напис «PLAY», а S2 – «STOP».

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Яке призначення функції *shiftIn ()*?
2. Поясніть термін «ущільнення виводів». Як та для чого його потрібно використовувати для семисегментного індикатора?
3. Як здійснювати обмін інформацію через інтерфейс SPI?
4. Поясніть призначення сигналів MISO, MOSI, SCK, SS?
5. Намалюйте схему підключення до Arduino Nano восьми розрядного семисегментного індикатора?

3.6 Практична робота №6. Робота з LCD – дисплеєм

Мета: ознайомитись з принципом роботи LCD – дисплея з контролером HD44780 у 4 бітному режимі підключення та дослідити можливості виведення на дисплей інформації; закріпити навички роботи з цифровими портами, тактовими кнопками, масивами.

Завдання: написати програму для виводу даних і керування LCD – дисплеєм з використанням тактових кнопок, АЦП лабораторного макету / віртуального стенду «Arduino Learner Kit» Arduino.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

На рис. 3.5 наводиться схема модуля LCD індикатора 16×2 з контролером

HD44780. До виводу Vo приєднується потенціометр (RV2), яким встановлюється контрастність зображення на LCD індикаторі.

Вивід RS використовується для вибору того, що буде надсилатися до індикатору (команди чи дані). Так якщо RS=0, то надсилаються команди на LCD індикатор (встановіть курсор на певне місце, очистити). Коли RS=1, то надсилаємо дані або символи.

Вивід R / W вибирає режим читання або запису LCD індикатора. Режим запису використовується для відправлення команд та даних до індикатора.

Вивід E дозволяє записувати до регістрів індикатора даних від D0 до D7.

Виводи A (анод) і K (катод) використовуються для світлодіодного підсвічування екрану індикатора через струмообмежувальний резистор R24 (220 Ом). LCD індикатор може використовувати 4 або 8-бітовий режим передачі даних. У лабораторному макеті використовується 4-бітний режим.

Для відображення даних на індикаторі необхідно підключити виводи RS, E, DB4-DB7 (з'єднувач X8) до Arduino та подати живлення на індикатор через перемикач SW1 (контакт 3 перевести у положення «On»).

Бібліотека LiquidCrystal дозволяє Arduino управляти різними LCD дисплеями, побудованими на базі поширеного чіпсета Hitachi HD44780 (або сумісного). У бібліотеці реалізований як 4-х, так і 8-бітний режим роботи. Бібліотека має такі функції: LiquidCrystal(), begin(), clear(), home(), setCursor(), write(), print(), cursor(), noCursor(), blink(), noBlink(), display(), noDisplay(), scrollDisplayLeft(), scrollDisplayRight(), autoscroll(), noAutoscroll(), leftToRight(), rightToLeft(), createChar().

LiquidCrystal (rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7) задає конфігурацію підключення та режим роботи, де *rs, rw, enable*: номер виводу Arduino, з'єднаного з виводом RS, RW, E LCD-індикатора; *d0, d1, d2, d3, d4, d5, d6, d7*: номера виводів Arduino, які підключені до відповідних цифрових виводів LCD-індикатора. Параметри rw, d0, d1, d2 і d3 є не обов'язковими. Якщо d0, d1, d2 і d3 не вказані, то LCD буде працювати у 4-х бітному режимі (d4, d5, d6, d7).

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
```

```
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

Приклад застосування

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
void setup(){
  lcd.begin(16,1);
  lcd.print("hello, world!");
}
void loop() {}
```

lcd.begin (cols, rows) ініціалізує інтерфейс для взаємодії з LCD-індикатором та задає розміри (ширину і висоту) області виведення екрану, де *lcd*: змінна типу LiquidCrystal, cols: кількість стовпців екрану, rows: кількість рядків екрану. При роботі з LCD-дисплеєм, функція *begin()* повинна викликатися першою і передувати іншим командам з бібліотеки LiquidCrystal.

lcd.clear () очищає LCD-екран і переміщує курсор в лівий верхній кут.

lcd.home () переміщує курсор в лівий верхній кут екрану (наступний текст буде виводиться з початку екрану).

lcd.setCursor (col, row) встановлює позицію, в якій буде виводитися наступний текст, де *col*: координата X позиції курсора (0 означає перший стовпець)**;** *row:* координата Y позиції курсора (0 означає перший рядок).

lcd.write (data) виводить символ на LCD-індикатор, де *data:* символ, який необхідно вивести на екран.

lcd.print(data) / lcd.print(data, BASE) виводить текст на LCD-індикатор, де *data*: дані, які необхідно вивести (тип char, byte, int, long або string); BASE (не обов'язковий параметр): основа системи числення, в якій необхідно виводити числа: BIN – двійкова, DEC – десяткова, OCT – вісімкова, HEX – шістнадцяткова.

lcd.cursor() / lcd.noCursor () показує/ не показує на LCD-екрані курсор: символ підкреслення в тій позиції, куди буде виведий наступний символ.

lcd.blink() / lcd.noBlink () включає / відключає на LCD-індикаторі курсор, що мигає.

lcd.noDisplay() / lcd.display() вимикає / вмикає LCD-екран. Текст, якщо не відображається на екрані, зберігається в пам'яті.

lcd.scrollDisplayLeft() / lcd.scrollDisplayRight() здійснює прокручування вмісту дисплея (весь текст і курсор) на один символ ліворуч / праворуч.

lcd.autoscroll() включає автоматичну прокрутку тексту на LCD. Це означає, що при виведенні кожного нового символу, всі попередні символи будуть зсуватись на одну позицію. Якщо встановлений режим перегляду тексту зліва-направо (за замовчуванням), то прокрутка буде здійснюватися ліворуч; якщо встановлений режим зправа-наліво, то прокрутка буде здійснюватися праворуч. Таким чином, кожен новий символ буде виводиться в одній і тій же позиції LCD.

lcd.noAutoscroll () функція відключає автоматичну прокрутку тексту в LCD.

lcd.leftToRight () / lcd.rightToLeft() встановлює режим перегляду тексту на LCD зліва-направо (режим за замовчуванням) / зправа-наліво.

lcd.createChar (num, data) створює символ користувача для LCD-індикатора. Дисплей підтримує до 8 символів користувача (пронумерованих від 0 до 7) розміром 5x8 пікселів. Зовнішній вигляд кожного символу користувача задається масивом з восьми байт, кожен з яких характеризує відповідний рядок. П’ять молодших біт кожного байта визначають стан пікселів у відповідному рядку. Для того, щоб вивести певний символ користувача, використовується функцію write () з його номером, де *num:* номер призначеного символу користувача, який необхідно створити (від 0 до 7); *data:* дані у пікселях символу користувача

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
byte smiley[8] = {
  B00000,
  B10001,
  B00000,
```

```
  B00000,
  B10001,
  B01110,
  B00000,
};
void setup() {
  lcd.createChar(0, smiley);
  lcd.begin(16, 2);
  lcd.write(byte(0));
}

void loop() {}
```

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму LCD1 (додаток Ж) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання Arduino та LCD індикатора у відповідності до програми. Дослідити роботу програми.

3. Завантажити програму LCD2 (додаток Ж) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

4. Завантажити програму LCD3 (додаток Ж) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

5. Завантажити програму LCD4 (додаток Ж) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

6. Завантажити програму LCD5 (додаток Ж) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

7. Завантажити програму LCD6 (додаток Ж) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

8. Завантажити програму LCD7 (додаток Ж) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання Arduino, LCD індикатора, потенціометра, світлодіода у відповідності до програми. Дослідити роботу програми.

Завдання

1. Реалізувати програму, яка виводить прізвище, ім’я та по батькові у вигляді рядка на LCD-індикаторі, що «біжить».

2. Внести зміни до попередньої програми, щоб виводилась ще поточна дата.

3. Реалізувати програму виведення випадкового числа на LCD-індикаторі за натисненням кнопки.

4. Реалізувати програму, яка відображає на LCD-індикаторі напис «LOAD». За натисненням кнопки S1 відбражається напис «PLAY», а S2 – «STOP».

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Які фукції присутні у бібліотеці LiquidCrystal?
2. У чому різниця 4 бітного та 8 бітного режиму роботи? Коли доцільно застосовувати кожний з режимів?
3. Як виконується ініціалізація роботи з LCD-індикатором?
4. Як використовуються сигнали RS, RW, E для роботи з LCD-індикатором?
5. Як вивести україномовний текст на LCD-індикатор?

3.7 Практична робота №7. Програмування Arduino. Дослідження роботи датчика температури LM35

Мета: ознайомитись з принципом роботи та зчитуванням даних датчика температури LM35; закріпити навички виведення інформації на семигментний індикатор з використанням регістру зсуву 74HC595 та LCD-індикатор.

Завдання: написати програму для зчитування та передачі значення температури до LED або LCD індикатора.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

LM35 - Прецизійний аналоговий датчик температури, на виході якого формується напруга пропорційно температурі за шкалою Цельсія.

Характеристики датчика:

- діапазон температур: - 55 ° C ... 150 ° C ± 0.5 ° C;
- дозвіл: 10.0 mV / ° C;
- напруга на виході при 25 ° C: 250мВ;
- напруга живлення: від 2,7 до 5,5 В.

На виході датчика формується напруга пропорційно температурі за шкалою Цельсія, величина напруги 10.0 мВ на 1 °C, тобто, якщо температура датчика 25°C на виході датчика буде 250 мВ.

Датчик LM35 має три виводи: плюс живлення (+2.7 - 5.5В), вихід, земля (рис. 3.18).

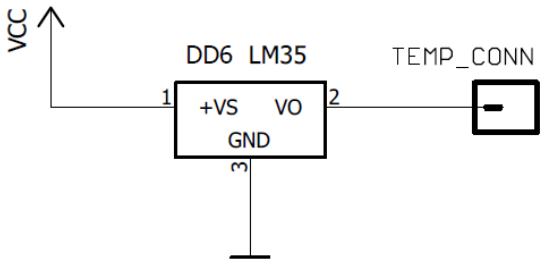


Рисунок 3.18 – Схема підключення датчика LM35

Додаткові бібліотеки для Arduino не потрібні. Приклад програми роботи з датчиком LM35:

```
int raw = 0;
float temp = 0;

void setup() {
    Serial.begin(9600);
    pinMode(A0, INPUT);
}

void loop() {
    raw = analogRead(A0);
    temp = (raw/1023.0)*5.0*1000/10;
    Serial.println(temp);
    delay(1000);
}
```

У програмі можна помітити вираз:

```
temp = (raw / 1023.0) * 5.0 * 1000/10;
```

Усі аналогові датчики мають важливу характеристику − відношення кількості вольт до одиниці вимірюваної величини. У датчика LM35 кожен градус вимірюваної температури, відповідає 10 мВ напруги на виході. Тому значення, що зчитано за допомогою analogRead, необхідно перетворити у вольти:

```
вольти = (значення АЦП / 1023) * 5
```

Така процедура називається нормуванням: 1023 − максимальне значення, яке може повернути 10-бітний АЦП, вбудований в Arduino; 5 − опорна напруга АЦП.

Далі перетворимо вольти в градуси Цельсія:

```
градуси = (вольти * 1000) / 10
```

Перетворюємо вольти в мілівольтах (* 1000), і ділимо на 10.

Неможливо здійснити вимірювання негативних температур, 0 °C це 0В на виході датчика. Щоб вимірювати весь діапазон потрібно подавати негативну напругу, але навіть якщо вона буде подана, вбудований АЦП в Arduino не може вимірювати негативну напругу. Низький дозвіл вбудованого АЦП Arduino та нестабільність опорної напруги у випадку якої використовується напруга живлення 5В. Вирішується використанням вбудованого в Arduino джерела опорного напруги 1,1В. У цьому випадку верхня межа температур, яку можно виміряти, буде 110 °C.

Датчик аналоговий і відповідно підключати його потрібно на аналоговий вхід Arduino. У прикладі коду, що навевдений нижче, датчик контактом TEMP_CONN (рис.3.1) підключений до входу A0.

```
float  tempC;
int  reading;

void  setup () {
  analogReference (INTERNAL);
  // включаємо внутрішне джерело опорної напруги 1,1В
  Serial.begin (9600);
}
```

```
void  loop () {
  reading  =  analogRead(A0);
  tempC  =  reading / 9.31; // переводимо градуси Цельсія
  Serial.print (tempC);
  Serial.println ("C");
  delay (1000);
}
```

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму LM35_SEG (додаток И) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання датчика LM35, семисегментного індикатора та Arduino у відповідності до програми. Дослідити роботу програми.

3. Завантажити програму LM35_Shift (додаток И) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання датчика LM35, семисегментного індикатора, регітру зсуву 74HC595 та Arduino у відповідності до програми. Дослідити роботу програми.

4. Завантажити програму LM35_LCD (додаток И) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання датчика LM35, LCD-індикатора та Arduino у відповідності до програми. Дослідити роботу програми.

Завдання

1. Реалізувати програму, яка виводить на LCD-індикатор значення температури з датчика LM35 та керує RGB світлодіодом. Якщо t >18°C, то світиться синій світлодіод; якщо t ≥ 25°C, то світиться зелений світлодіод; якщо t ≥ 33°C, то світиться червоний світлодіод; якщо t ≤ 18°C, то RGB світлодіод не світиться.

2. Реалізувати програму, яка виводить на семисегментний індикатор значення температури з датчика LM35 та керує RGB світлодіодом. Якщо t >18°C, то світиться синій світлодіод; якщо t ≥ 25°C, то світиться зелений світлодіод;

якщо t ≥ 33°C, то світиться червоний світлодіод; якщо t ≤ 18°C, то RGB світлодіод не світиться.

3. Реалізувати програму, яка виводить на семисегментний індикатор з використанням регістру зсуву 74HC595 значення температури з датчика LM35 та керує RGB світлодіодом. Якщо t >18°C, то світиться синій світлодіод; якщо t ≥ 25°C, то світиться зелений світлодіод; якщо t ≥ 33°C, то світиться червоний світлодіод; якщо t ≤ 18°C, то RGB світлодіод не світиться..

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Поясніть термін нормування?
2. Як виміряти від'ємну температуру датчиком LM35?
3. Що виконує команда analogReference (INTERNAL)?
4. Чому використовують опорну напругу 1,1В, а не 5В для АЦП Arduino?
5. Які є аналоги датчика LM35?
6. Які є рекомендації до застосуванню датчика LM35?

3.8 Практична робота №8. Програмування Arduino. Дослідження роботи датчика температури та вологості DHT11

Мета: ознайомитись з принципом роботи та зчитуванням даних датчика температури та вологості DHT11; закріпити навички виведення інформації на семигментний індикатор з використанням регістру зсуву 74HC595 та LCD-індикатор.

Завдання: написати програму для зчитування та передачі значення температури та вологості до LED або LCD індикатора.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

Датчик DHT11 – це цифровий датчик температури і вологості, що

дозволяє калібрувати цифровий сигнал на виході. Складається з ємнісного датчика вологості та термістора. Також, датчик містить в собі АЦП для перетворення аналогових значень вологості та температури.

Характеристики:

- визначення вологості: 20-90% RH ± 5% (макс.);
- визначення температури: 0-50 ºС ± 2% (макс.);
- частота опитування: не більше 1 Гц;
- розміри 15.5 × 12 × 5.5 мм;
- 4 виводи з відстанню між контактами 2,54 мм;
- живлення 3.5 – 5.5 В.

Виводи:

1. VDD (живлення).
2. Data Out – вивід даних.
3. NC – не використовується.
4. Загальний.

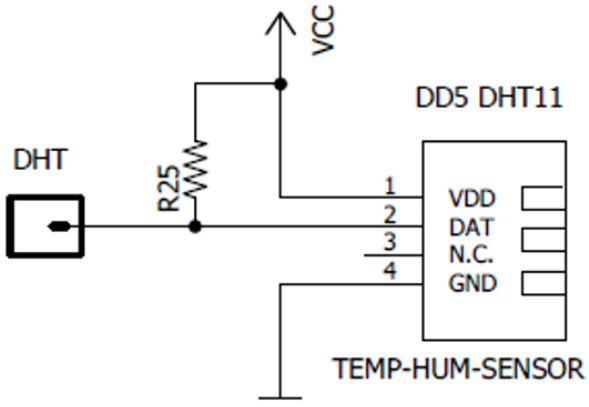


Рисунок 3.19 – Схема підключення датчика DHT11

Для роботи з датчиком використовують клас DHT, який містить декілька функцій:

DHT. begin() ініціалізує роботу датчика.

DHT.readTemperature(bool Scale) вимірює температуру повітря, де *Scale*: false – температура по Цельсію, true – температура по Фаренгейту; значення, що повертається: temp (float): температура.

DHT.convertFtoC(float temp) перетворює значення температури по Фаренгейту в температуру по Цельсію, де temp – температура по Фаренгейту; значення, що повертається: температура по Цельсію.

DHT.convertCtoF(float temp) перетворює значення температури по Цельсію в температуру по Фаренгейту, де temp – температура по Цельсію; значення, що повертається: температура по Фаренгейту.

DHT.readHumidity() вимірює вологість повітря; значення, що повертаються: hum (float): вологість.

Скетч термодатчика DHT11 для Ардуіно:

```
#include <DHT.h>      // підключаємо бібліотеку для датчика
DHT dht (2, DHT11);  // повідомляємо на якому порту буде датчик

void  setup () {
   dht.begin ();            // запускаємо датчик DHT11
   Serial.begin (9600);   // підключаємо монітор порту
}

void  loop () {
    // зчитуємо температуру (t) і вологість (h)
   float h = dht.readHumidity();
   float t = dht.readTemperature();

   // виводимо температуру (t) і вологість(h) на монітор порту
   Serial.print ("Humidity:");
   Serial.println (h);
   Serial.print ("Temperature:");
   Serial.println (t);
}
```

При підключенні до мікроконтролера, між виводами VDD і Data включають «pull-up» резистор номіналом 10 кОм (рис. 3.19). Плата Arduino має вбудовані «pull-up» резистори, однак вони дуже слабкі – близько 100 кОм.

На рис.3.7 наведена схема розміщення елементів лабораторного макету «Arduino Learner Kit». Контакт DHT призначений для зчитування інформації з датчика DHT11.

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму DHT11_SEG (додаток К) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання датчика DHT11, семисегментного індикатора та Arduino у відповідності до програми. Встановити бібліотеку DHT.h в середовище Arduino IDE (див. п. 1.3). Дослідити роботу програми.

3. Завантажити програму DHT11_LCD (додаток К) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання датчика DHT11, LCD-індикатора та Arduino у відповідності до програми. Дослідити роботу програми.

Завдання

1. Реалізувати програму, яка виводить на LCD-індикатор значення температури та вологості з датчика DHT11 та керує RGB світлодіодом. Якщо t >18°C, то світиться синій світлодіод; якщо t ≥ 25°C, то світиться зелений світлодіод; якщо t ≥ 33°C, то світиться червоний світлодіод; якщо t ≤ 18°C, то RGB світлодіод не світиться.

2. Реалізувати програму, яка виводить на семисегментний індикатор значення температури та вологості з датчика DHT11 та керує RGB світлодіодом. Якщо вологість h < 40%, то світиться синій світлодіод; якщо 60% ≥ h ≥ 40°%, то світиться зелений світлодіод; якщо h > 60%, то світиться червоний світлодіод.

3. Реалізувати програму, яка виводить на семисегментний індикатор з використанням регістру зсуву 74HC595 значення температури та вологості з датчика DHT11 та керує RGB світлодіодом. Якщо вологість h < 40%, то світиться синій світлодіод; якщо 60% ≥ h ≥ 40°%, то світиться зелений світлодіод; якщо h > 60%, то світиться червоний світлодіод.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань)

Контрольні питання

1. Призначення та підключення датчика DHT
2. Опишіть формат передачі даних з датчика DHT
3. Який інтерфейс використовується для передачі даних?
4. Опишіть основні функції класу DHT.
5. Який алгоритм роботи має програма для зчитування показників температури і вологості?
6. Чим відрізняються датчики DHT11 та DHT22? Як налаштувати програму DHT11_LCD для роботи з датчиком DHT22?

3.9 Практична робота №9. Робота з інтерфейсом TWI (I^2C) та годинником реального часу DS1307

Мета: ознайомитись з принципом роботи інтерфейсом TWI (I^2C) Arduino та обіну даними з годинником реального часу DS1307; закріпити навички виведення інформації на семигментний індикатор з використанням регістру зсуву 74HC595 та LCD-індикатор.

Завдання: написати програму годинника / будильника з відображенням інформації на LED або LCD індикаторі.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з’єднувачі.

Теоретичні відомості

Двопроводовий послідовний інтерфейс TWI ідеально підходить для типових додатків на мікроконтролерах і вимагає тільки дві лінії зв'язку. Протокол TWI дозволяє проектувальнику системи зовні зв'язати до 128 різних пристроїв через одну двухпроводную двосторонню шину, де одна лінія – лінія синхронізації SCL і одна – лінія даних SDA. В якості зовнішніх апаратних компонентів, які потрібні для реалізації шини, потрібен лише підтягуючий до плюса живлення резистор на кожній лінії шини. Всі пристрої, які підключені до шини, мають свої індивідуальні адреси.

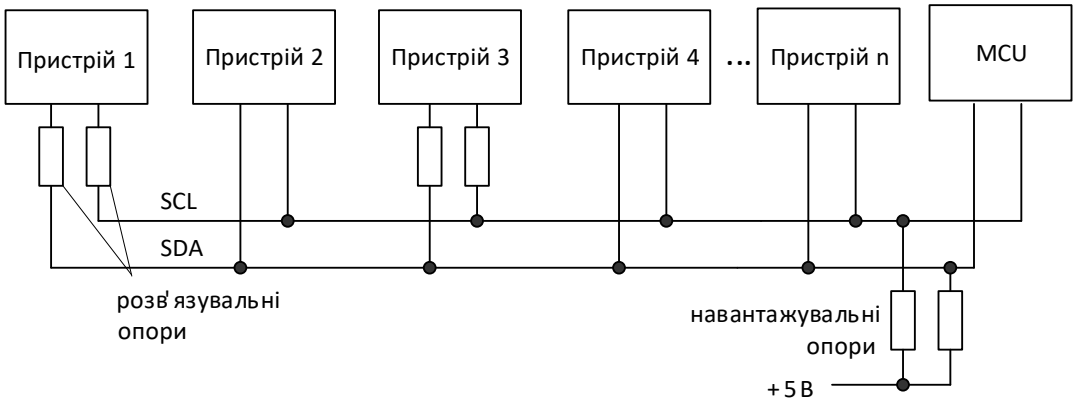


Рисунок 3.20 – Система протоколу TWI

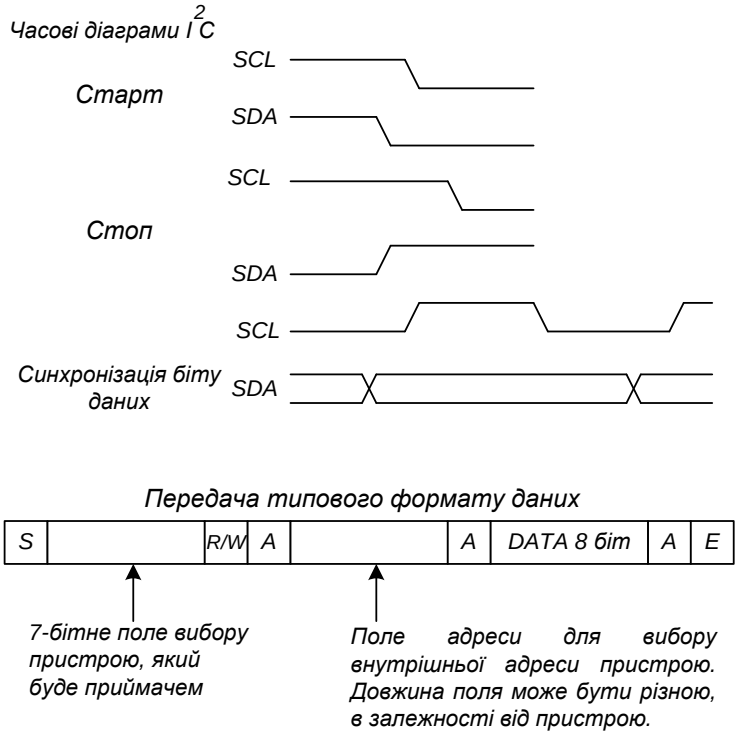


Рисунок 3.21 – Часові діаграми роботи шини I^2C

Як показано на рис. 3.20, обидві лінії шини підключені до шини живлення через навантажувальні (pull up) резистори. У всіх сумісних з TWI пристроях, як драйвер шини, використовуються транзистор або з відкритим стоком, або з відкритим колектором. Так реалізована функція, яка дуже важлива для двобічної роботи інтерфейсу. Низький логічний рівень на лінії шини TWI генерується, якщо один або більше з TWI-пристроїв виводить логічний 0. Високий рівень на лінії присутній, якщо всі TWI-пристрої перейшли у третій високоімпедансний стан, дозволяючи підтягуючим резисторам задати рівень логічної 1. Зверніть увагу, що при підключенні до шини TWI декількох AVR- мікроконтролерів, для

роботи шини, усі ці мікроконтролери повинні бути підключені до живлення.

Кількість пристроїв, яка може бути підключено до однієї шини обмежується гранично допустимої ємністю шини (400 пФ) і 7-розрядним простором адрес. Підтримуються два різних набори технічних вимог, де один набір - для шин зі швидкістю передачі даних нижче 100 кГц, а інший дійсний для швидкостей понад 400 кГц.

Уся передача даних складається із стартової посилки, бітів і стопової посилки. Початок передачі визначається Start послідовністю - провал SDA при високому рівні SCL (рис. 3.21).

При передачі інформації від Master до Slave, Master генерує такти на SCL і видає біти на SDA. Які Slave зчитує коли SCL стає 1.

При передачі інформації від Slave до Master, Master генерує такти на SCL і дивиться, що там Slave робить з лінією SDA – зчитує дані. А Slave, коли SCL йде в 0, виставляє на SDA біт, який Master зчитує коли підніме SCL назад.

Закінчується все STOP послідовністю. Коли при високому рівні на SCL лінія SDA переходить з низького на високий рівень.

Перший пакет від Master до Slave – це фізична адреса пристрою і біт напрямку (рис. 3.21).

Сама адреса складається з семи біт (ось чому до 127 пристроїв на шині), а восьмий біт означає, що буде робити Slave на наступному байті – приймати або передавати дані. Дев'ятим бітом йде біт підтвердження ACK. Якщо Slave розпізнав свою адресу повністю, то на дев'ятому такті він переведе лінію SDA в 0, згенерувавши ACK – тобто зрозумів. Тоді Master продовжить передачу даних. Якщо Slave не відповів, тобто SDA на дев'ятому такті не буде переведено в 0 (не буде ACK), то майстер припинить свої спроби підєднатися.

Після адресного пакета йдуть пакети з даними в ту або іншу сторону, в залежності від біта R/W в заголовному пакеті.

На Arduino UNO, Nano, Pro Mini виводи I^2C: контакт SDA – A4, контакт SCL – A5.

Бібліотека Wire.h дозволяє Arduino взаємодіяти з різними пристроями по інтерфейсу I^2C / TWI. Згідно з протоколом I2C, адреса пристрою може складатися як з 7, так і з 8 біт. Як правило, 7 біт ідентифікують пристрій, в той час, як восьмий біт задає напрям передачі даних: від пристрою (читання) або до нього (запис). Всі функції бібліотеки Wire.h використовують 7-бітну адресацію, тим самим обмежуючи діапазон можливих адрес в межах 0 – 127.

Функції I^2C / TWI на платах Arduino:

Wire.begin(address) ініціалізує бібліотеку Wire і підключає Arduino до шини I^2C в ролі ведучого (master) або веденого (slave) пристрою, де *address*: 7-бітова адреса Slave-пристрою (необов'язковий параметр); якщо адреса не вказана, то Arduino виступає в ролі Master-пристрою.

Wire.requestFrom (address, quantity) запрошує дані у веденого пристрою (slave); як правило, використовується тільки ведучим пристроєм (Master). Після виклику requestFrom () запитувані дані повинні бути зчитані за допомогою функцій available () і read (), де *address*: 7-бітова адреса відомого пристрою, у якого запитуються дані; *quantity*: кількість запитуваних байт.

Wire.beginTransmission (address) починає процедуру передачі даних по інтерфейсу I^2C веденому пристрою з вказаною адресою. Для подальшої відправки даних, необхідно спершу поставити їх в чергу за допомогою функції write (), після чого здійснити, безпосередньо, передачу функцією endTransmission ().

Wire.endTransmission () завершує процедуру передачі даних веденому пристрою, ініційовану функцією beginTransmission(). При цьому функція відправляє байти, поставлені в чергу функцією write ().

Wire.write(value) Wire.write(string) Wire.write(data, length) повертає кількість записаних байт, де value: значення, яке необхідно відправити у вигляді одиночного байта; string: рядок, який необхідно відправити у вигляді послідовності байт; data: масив даних, який необхідно відправити у вигляді декількох байт; length: кількість переданих байт.

Wire.available () повертає кількість байт, доступних для зчитування

функцією read (). На ведучому пристрої (Master), ця функція має викликатися після функції requestFrom (), а на веденому (Slave) - всередині обробника onReceive ().

Wire.read () зчитує байт даних, отриманий ведучим пристроєм від веденого (або навпаки) в результаті виконання функції requestFrom ().

DS1307 – це годинник реального часу з екстремально точним ходом, завдяки вбудованому кварцовому резонатору з температурною компенсацією. Інтерфейс передачі даних – I^2C. У мікросхемі є також вхід для підключення резервної батареї (рис. 3.6). При відключенні основного живлення мікросхема автоматично перемикається на роботу від резервної батареї, точність ходу від резервної батареї не порушується.

У DS1307 підтримується підрахунок секунд, хвилин, годин, днів місяця (дати), днів тижня, місяців і років (з урахуванням високосного року для місяців). Підтримується робота в 12 і 24 годинному форматі.

Для підключення RTC годинника реального часу DS1307 було розроблено декілька бібліотек: Wire.h, TimeLib, DS1307RTC.h, DS1307.h, iarduino_RTC.h. Варто звернути увагу на iarduino_RTC.h. Бібліотеки універсальні (підходить для DS3231, DS1302). У додатку Л наведені приклади роботи з DS1307 та виводом інформації на LED та LCD індикатор. У прикладах використовується бібіліотека DS1307.h, яку потрібно встановити на комп‘ютер або додати в директорію з файлом проекту. Використовуються такі функції бібіліотеки:

DS1307.begin() – ініціалізація роботи RTC модуля.

DS1307.getDate(clock) – отримання часу.

DS1307.setDate (Р, М, Д, ДТ, Г, Х, С) – встановлення часу (рік, місяць, день, день тижня, години, хвилини, секунди).

Хід виконання роботи

1. Підключити схему до комп’ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму DS1307_LCD (додаток Л) до лабораторного

макета / віртуального стенду, попередньо виконати з'єднання DS1307, LCD-індикатора, тактових кнопок та Arduino у відповідності до програми. Встановити бібліотеку DS1307.h в середовище Arduino IDE (див. п. 1.3) Дослідити роботу програми.

3. Завантажити програму DS1307_SEG (додаток Л) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання DS1307, LCD-індикатора, тактових кнопок та Arduino у відповідності до програми. Дослідити роботу програми.

Завдання

1. Установити поточне значення часу на годинник. Реалізувати програму, яка виводить на LCD-індикатор дату, час, прізвище автора та виводить ці значення в Монітор послідовного порту кожні 5 сек.

2. Реалізувати програму, яка виводить на LCD-індикатор поточний час та час будильника. Виставити значення часу та будильник. При спрацьовуванні будильника звучить тональний сигнал та загорається світлодіод.

3. Реалізувати програму, яка виводить на семисегментний індикатор час та дату (по черзі) з можливістю установки дати та часу.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань).

Контрольні питання

1. Що таке DS1307, які має характеристики та який інтерфейс використовує?
2. Коротко опишіть принцип підключення годинника до Arduino.
3. Які бібліотеки існують для роботи DS1307?
4. Які основні функції має, описана в роботі, бібліотека DS1307?
5. Які значення та в яких діапазонах повертаються годинником?
6. Особливості обміну інформацією по шині I^2C.
7. Навести часові діаграми роботи шини I^2C для передавання інформації від Arduino до DS1307.

3.10 Практична робота №10. Робота з матричним світлодіодним індикатором

Мета: ознайомитись з принципом роботи матричного світлодіодного індикатора 8×8 точок, драйвера керування роботою світлодіодною матрицею MAX7219; навчитись програмувати виведення інформації через інтерфейс SPI на матричний світлодіодний індикатор з використанням драйвера керування MAX7219.

Завдання: написати програму представлення символів та зображень на матричному світлодіодному індикаторі 8×8.

Обладнання: лабораторний макет/віртуальний стенд «Arduino Learner Kit»; USB – кабель; провідники-з'єднувачі.

Теоретичні відомості

Матричні світлодіодні індикатори (МСІ) використовуються для відображення алфавітно-цифрової інформації. Кожен з таких МСІ, виконаний у вигляді інтегральної мікросхеми, є матрицею світлодіодів розмірністю *m* x *n*, де *n* - число стовпчиків, *m* - число рядків матриці. Найбільшого поширення набули МСІ з розмірністю матриці 7 × 5, 9 × 7, 8 × 8 (рис. 3.22).

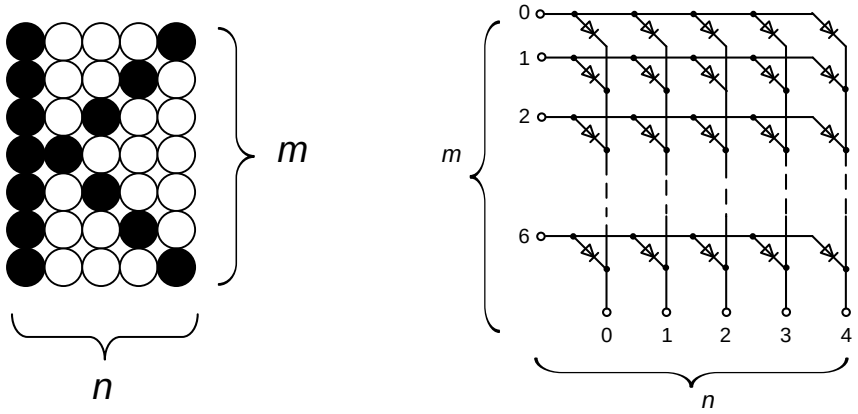


Рисунок 3.22 – Загальний вигляд та схема матричного індикатора

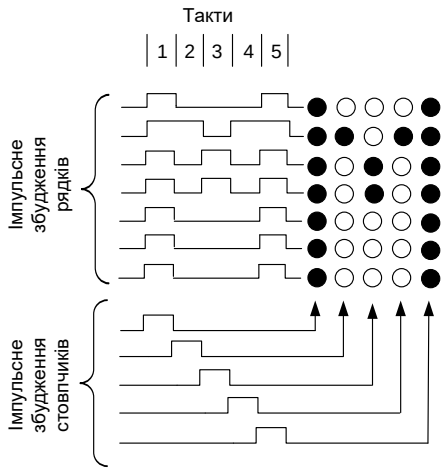


Рисунок 3.23 – Часова діаграма формування літери М

У кожному часовому такті збуджується строб імпульс відповідного стовпця. У результаті відбувається відображення інформації у всіх елементів даного стовпця. Після кожного такту відбувається зсув інформації і в наступному часовому такті збуджується строб імпульс у другому стовпчику і так далі. За п'ять тактів відбувається передача повної інформації на матричний індикатор, після чого відбувається повторення передачі, якщо по шині введення даних не поступила нова інформація. Часова діаграма формування букви М представлена на рис. 3.23.

У лабораторному макеті для представлення символів та зображень використовується світлодіодна матриця 8×8 точок (64 світлодіоди). Вона має 16 виводів для управління рядками та стовпцями масиву. Спеціалізована мікросхема-драйвер MAX7219 (рис. 3.4) призначена для керування світлодіодною матрицею 8×8. MAX7219 — це 16-бітний регістр послідовного зсуву. Перші 8 біт задають команду, а решта 8 біт використовуються для визначення даних для команди. На 18 ніжку Iset підключається резистор «pull up», який встановлює піковий струм для сегментів. Живлення на мікросхему MAX7219 подається через перемикач SW1 (контакт 4 перевести у положення «On»). Виводи DIN (вхід даних), CLK (вхід для сінхроімупульсів) і CS (вхід вибору мікросхеми) через X1 з’єднують з цифровими портами плати Arduino.

Управляти світлодіодним матрицею можна не тільки самостійно, але і за

допомогою різних бібліотек. Одна з таких бібліотек **LedControl.h.**

Для роботи з бібліотекою необхідно створимо об'єкт класу LedControl . Типовий код ініціалізації бібліотеки буде виглядати так:

```
#include "LedControl.h"
LedControl LC = LedControl(12, 11, 10, 1);
```

При підключенні MAX72xx можна використовувати будь-які порти на платі Arduino, але так як є входи / виходи які використовуються для послідовного з'єднання (порти 0 та 1), а також порти які мають у своїй зв'язці світлодіодний індикатор (порт 13), то краще уникати підключення до цих портів вибравши будь-які інші доступні. Також, порти зазначені в оголошенні об'єкта класу, не потрібно форматувати функцією *pinMode()* в розділі *setup ()* програми, бібліотека LedControl при створенні об'єкта класу сама проініціалізірует ці порти потрібним чином.

Четвертий параметр є кількістю пристроїв з драйвером MAX72xx підключених до однієї шині каскадом. Одному об'єкту класу LedControl можна адресувати до 8 пристроїв, при цьому, чим більше пристроїв на шині, тим відповідно нижче її продуктивність. Дозволені тільки значення від 1 до 8 включно, оскільки одному об'єкту класу LedControl можна адресувати понад 8-ми пристроїв. Якщо необхідно управляти більш вісьмома пристроями на базі драйвера MAX72xx , то можна створити ще кілька об'єктів класу LedControl . Для цього потрібно використовувати іншу групу контактів підключення пристрою, відмінну від першого об'єкта класу LedControl . Наприклад ось так:

```
#include "LedControl.h"
LedControl LC1 = LedControl(12, 11, 10, 8);
LedControl LC2 = LedControl(9, 8, 7, 8);
```

При ініціалізації пристрою на базі драйвера MAX72xx (за умови підключення всього лише одного пристрою), можна використовувати таку конструкцію коду в розділі програми setup ():

```
void setup() {
  // Відключення режиму енергозбереження
  // Функція shutdown()
  LC1.shutdown(0, false);
  //Встанволення яскравості світіння світлодіодів у матриці
  LC1.setIntensity(0, 8);
  // Очистка матриці
  LC1.clearDisplay(0);
}
```

Функція ***shutdown()*** −це функція відключення режиму енергозбереження, світлодіоди споживають багато енергії, це може виявитися критичним, якщо пристрій працює від батарейок. Функція *shutdown ()* може відключити матрицю коли потрібно буде перевести пристрій в енергозберігаючий режим, або включити. *LC1.shutdown (int address, bool set),* де: *LC1* об'єкт класу LedContro*l, int address* адреса пристрою на базі драйвера MAX72xx, *bool set* вказує режим роботи (*false* − вихід з режиму очікування та відображення даних, *true* − пристрій переходить в сплячий режим).

Функція ***setIntensity()*** встановлює яскравість світіння сегментів, або світлодіодів (дивлячись що підключено дисплей або матриця). Прототип виклику функції: *LC1.setIntensity (int address, int brightness),* де: *LC1* об'єкт класу LedControl, *int address* адреса пристрою на базі драйвера MAX72xx, *int brightness* встановлює яскравість світіння сегментів, або світлодіодів. Може приймати значення від 0 до 15 (0 мінімальний рівень, 15 максимальний рівень).

Функція ***clearDisplay()*** очищає матрицю, за вказаною адресою.Прототип виклику функції: *LC1.clearDisplay (int address),* де: *LC1* об'єкт класу LedControl, *int address* адреса пристрою на базі драйвера MAX72xx.

Управління LED матрицями 8×8 здійснюється трьома функціями: setRow(), setColumn().

setLed() управляє кожним світлодіодом на матриці індивідуально. *LC1.setLed (int address, int row, int column, boolean state),* де: *LC1* об'єкт класу LedControl, *int address* адреса пристрою на базі драйвера MAX72xx, *int row* є ряд світлодіодів LED матриці 8×8, *int column* є стовпець світлодіодів LED матриці

8×8, *boolean state* це стан світлодіода (*true* − включений, *false* − виключений).

Приклад скетчу:

```
#include "LedControl.h"
LedControl LC = LedControl(12, 11, 10, 5);
void setup() {
  LC.shutdown(0, false);
  LC.setIntensity(0, 8);
  LC.clearDisplay(0);
}
void loop() {
   LC.setLed(0, 2, 7, true);
   delay(500);
   LC.setLed(0, 2, 7, false);
   delay(500);
}
```

Для включення ряду світлодіодів на LED матриці 8×8 застосовується функція *setRow ()*.Прототип виклику функції: *LC.setRow (int address, int row, byte value),* де: *LC1* об'єкт класу LedControl, *int address* адреса пристрою на базі драйвера MAX72xx, *int row* є ряд світлодіодів LED матриці 8×8, *byte value* − змінна типу *byte*, значення якої буде включати певні світлодіоди в ряду LED матриці 8×8.

Приклад скетчу:

```
#include "LedControl.h"
LedControl LC = LedControl(12, 11, 10, 5);
void setup() {
  LC.shutdown(0, false);
  LC.setIntensity(0, 8);
  LC.clearDisplay(0);
}
void loop() {
   LC.setRow(0, 2, B10110000);
}
```

Для включення стовців світлодіодів на LED матриці 8×8 застосовується функція *setColumn ()*.Прототип виклику функції: *LC.setColumn (int address, int column, byte value),* де: *LC1* об'єкт класу LedControl, *int address* адреса пристрою

на базі драйвера MAX72xx, *int column* є стовпець світлодіодів LED матриці 8×8, *byte value* − змінна типу *byte*, значення якої буде включати певні світлодіоди в ствпці LED матриці 8×8.

Приклад скетчу:

```
#include "LedControl.h"
LedControl LC = LedControl(12, 11, 10, 5);
void setup() {
  LC.shutdown(0, false);
  LC.setIntensity(0, 8);
  LC.clearDisplay(0);
}
void loop() {
   LC.setColumn (0, 0, B00000111);
}
```

Хід виконання роботи

1. Підключити схему до комп'ютера через USB порт плати Arduino та/або запустити віртуальний стенд у середовищі Proteus 8.

2. Завантажити програму *Matrix_One* (додаток М) до лабораторного макета / віртуального стенду, попередньо виконати з'єднання MAX7219 та Arduino у відповідності до програми. Встановити бібліотеку LedControl.h в середовище Arduino IDE (див. п. 1.3). Дослідити роботу програми.

3. Завантажити програму Matrix_Smile (додаток М) до лабораторного макета / віртуального стенду. Дослідити роботу програми.

Завдання

1. Реалізувати програму, яка передбачає використання світлодіодної матриці. Засвічуються стовпці матриці справа-наліво, потім – рядки згори-донизу, створюючи ефект роботи сканера.

2. Реалізувати програму, яка передбачає використання світлодіодної матриці. Засвічується крапка, яка «бігає» по контуру матриці.

3. Реалізувати програму, яка виводить на матрицю числа від 0 до 9, з кожним натисненням тактової кнопки.

4. Реалізувати програму, яка виводить на матрицю текст «Я ❤ ВНАУ».

5. Реалізувати програму, яка передбачає використання світлодіодної матриці. У центрі світлодіодної матриці засвічується крапка. За допомогою клавіш S0−S3 можна змінювати положення крапки вліво, вправо, вгору та вниз.

Підготувати звіт згідно ДСТУ 3008-95 (лістинг програми, висновки, перелік посилань).

Контрольні питання

1. Поясніть особливості обміну інформацією між мікросхемами інтерфейсом SPI?
2. Наведіть схему підключення 4 матричних світлодіодних індикаторів з використанням одного драйвера MAX7219. Поясніть як реалізувати програму, щоб текст рухався. Які функції бібліотеки LedControl.h будуть використовуватись?
3. Наведіть схему підключення 8 розрядного семисегментного індикатора з використанням одного драйвера MAX7219. Поясніть як реалізувати програму, щоб на індикаторі відображалась поточна дата та час. Які функції бібліотеки LedControl.h будуть використовуватись?
4. Які є візуальні редактори для створення анімаційних ефектів є? Поясніть як ними користуватись.

ЛІТЕРАТУРА

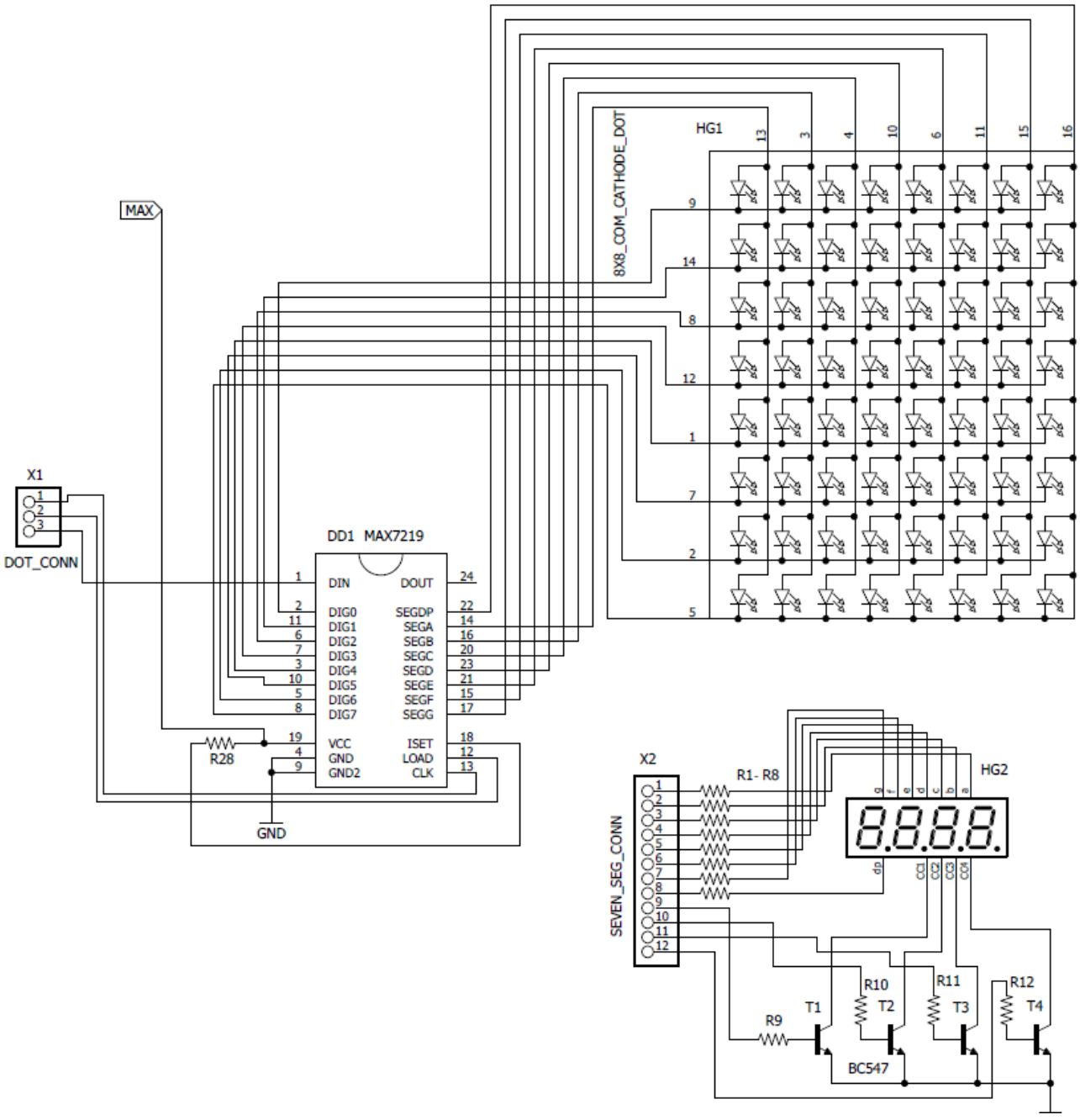
1. Цирульник С. М., Азаров О. Д., Крупельницький Л. В., Трояновська Т. І. Мікропроцесорна техніка: навчальний посібник. Вінниця: ВНТУ, 2017. 123 с.
2. Цирульник С. М. Лисенко Г. Л. Проектування мікропроцесорних систем. Вінниця: ВНТУ, 2012. 191 с.
3. Краткий учебный курс PROTEUS Руководство для начинающих. URL: http://proteus123.narod.ru.
4. Максимов А. Моделирование устройств на микроконтроллерах с помощью программы ISIS из пакета PROTEUS VSM. Радио. 2005. № 4, 5, 6. С. 30-33, 31-34, 30-32.
5. Мигаем светодиодом с помощью Ардуино эмулятора Tinkercad. URL: https://arduinoplus.ru/arduino-emulyator.
6. Autodesk Tinkercad Simulation of Arduino UNO Ping Pong Game V2.0. URL: https://www.instructables.com/id/Autodesk-Tinkercad-Simulation-of-Arduino-UNO-Ping.
7. Обзор приложения UnoArduSim v1.5.1. URL: https://kolotushkin.com/article.php?id=2.
8. Эмулятор Arduino UnoArduSim позволяет тестировать код без аппаратных средств. URL: https://cutt.ly/Qf2cIiB.
9. Рюмик С. М. 1000 и одна микронтроллерная схема. Вып. I. М.: Додэка-XXI, 2010. 356 с.
10. Рюмик С. М. 1000 и одна микронтроллерная схема. Вып. II. М.: «Додека-XXI», 2011. 400 с.
11. Офіційний сайт Arduino. URL: https://www.arduino.cc.
12. Сайт Arduino.ua Плати Arduino Nano. URL: https://doc.arduino.ua/ru/hardware/Nano
13. Справочник языка Ардуино. URL: https://doc.arduino.ua/ru/prog.
14. Полный список команд языка Ардуино. URL:

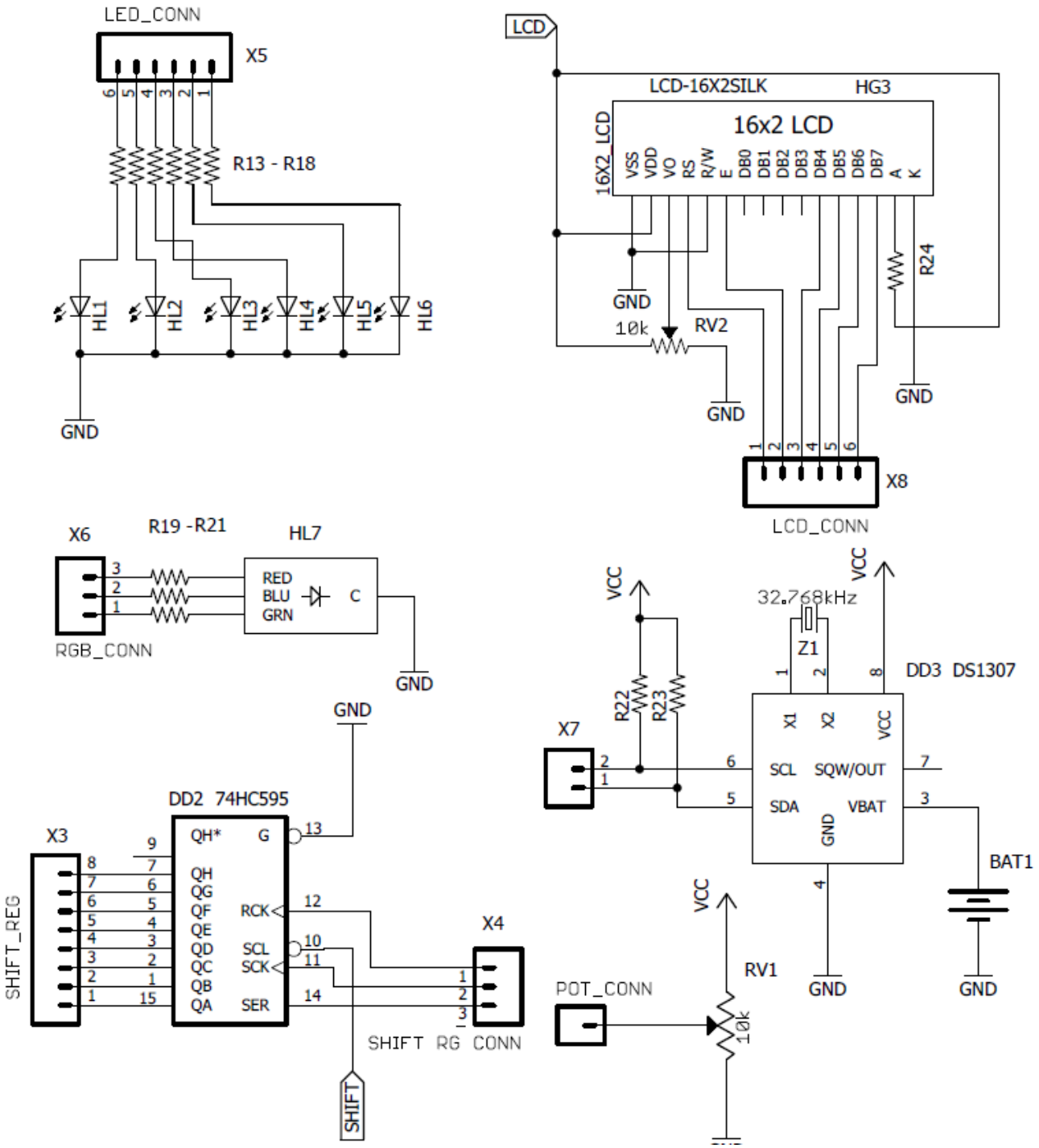
https://alexgyver.ru/lessons/arduino-reference.

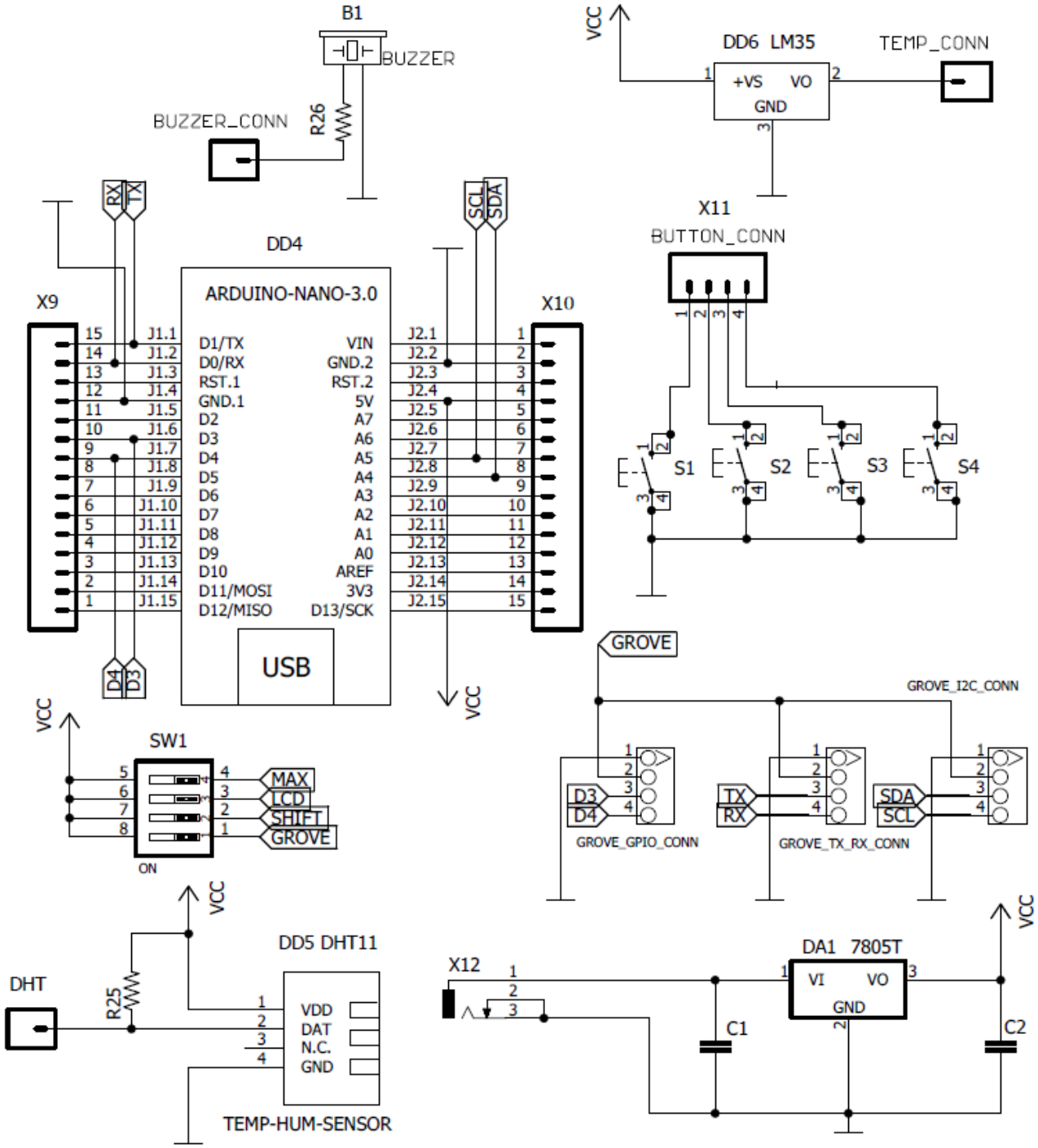
15. Программы для программирования плат Arduino и Digispark. URL: https://kolotushkin.com/indexsoft.php.
16. Библиотека LiquidCrystal. URL: http://developer.alexanderklimov.ru/arduino/liquidcrystal.php
17. LED-матрица 8×8. URL: http://developer.alexanderklimov.ru/arduino/ledmatrix.php
18. Визуальный редактор для создания анимационных эффектов LED Matrix Editor. URL: https://xantorohara.github.io/led-matrix-editor.

Додаток А

Лабораторний макет «Arduino Learner Kit». Схема електрична принципова




LED_CONN
X5
R13 - R18
HL1
HL2
HL3
HL4
HL5
HL6
GND
LCD
LCD-16X2SILK
HG3
16x2 LCD
16X2_LCD
VSS
VDD
VO
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
K
R24
GND
10k
RV2
GND
GND
X8
LCD_CONN
X6
R19 -R21
HL7
RED
BLU
GRN
C
RGB_CONN
GND
VCC
VCC
32.768kHz
Z1
DD3 DS1307
R22
R23
X7
X1
X2
VCC
SCL
SQW/OUT
SDA
VBAT
GND
BAT1
GND
GND
GND
DD2 74HC595
X3
SHIFT_REG
QH*
G
QH
QG
QF
QE
QD
QC
QB
QA
SER
RCK
SCL
SCK
X4
SHIFT RG CONN
SHIFT
VCC
RV1
POT_CONN
10k


B1
BUZZER
BUZZER_CONN
R26
VCC
DD6 LM35
TEMP_CONN
+VS
VO
GND
RX
TX
SCL
SDA
DD4
X11
BUTTON_CONN
X9
ARDUINO-NANO-3.0
X10
D1/TX
D0/RX
RST.1
GND.1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11/MOSI
D12/MISO
VIN
GND.2
RST.2
5V
A7
A6
A5
A4
A3
A2
A1
A0
AREF
3V3
D13/SCK
USB
S1
S2
S3
S4
GROVE
GROVE_I2C_CONN
SW1
MAX
LCD
SHIFT
GROVE
ON
GROVE_GPIO_CONN
GROVE_TX_RX_CONN
DHT
R25
DD5 DHT11
VDD
DAT
N.C.
GND
TEMP-HUM-SENSOR
X12
DA1 7805T
VI
VO
C1
C2

Додаток Б

Програма LED1

Програма демонструє роботу з цифровими портами Arduino. Алгоритм програми простий – загораються світлодіоди (HL1, HL4), пауза 0,5 секунди, світлодіоди гаснуть, пауза, потім процедура повторюється.

Лістинг програми LED1

```
int led1 = 2;
int led2 = 5;

void setup(){
  pinMode(led1, OUTPUT);    // порт як вихід
  pinMode(led2, OUTPUT);
}
void loop()
{
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
  delay (500);
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  delay (500);
}
```

Програма LED2

Програма демонструє роботу з регістрами портів Arduino. Алгоритм програми простий – світлодіоди відображають двійковий код від 3 до 256. Молодші біти порту D0, D1 не використовуються, тому програма використовує HL1-HL6 для відображення двійкового коду з паузою в 0,5 секунд.

Лістинг програми LED2

```
void setup()
{
  DDRD=0xFF;
  PORTD=0x00;
}
void loop(){
//PD2-PD7, PD0, PD1 не використовуються
 for (int i=3; i<256; i++)
  {
    PORTD=i;
    delay (500);
  }
}
```

Додаток В

Програма Tone

Програма демонструє формування звукового сигналу Arduino. Алгоритм програми простий – аналізується стан тактової кнопки. Якщо вона натиснута, то формується 3-х тональний звуковий сигнал з паузою 0,3 секунди.

Лістинг програми Tone

```
int buttonState = 0;
void setup()
{
    pinMode(2, INPUT_PULLUP);
    pinMode(3, OUTPUT);
}
void loop()
{
    buttonState = digitalRead(2);
    if (buttonState == LOW)
    {
        //tone(pin, frequency, duration)
        tone(3, 923, 300);
        delay(300);
        tone(3, 323, 300);
        delay(300);
        tone(3, 523, 300);
        delay(300);
    }
    else
    {
      noTone(3);
     }
}
```

Програма LED3

Програма демонструє застосування зовнішнього переривання INT1. Алгоритм програми простий – аналізується стан тактової кнопки. Якщо вона натиснута, то виникає переривання. Підпрограма переривання змінює стан світлодіода на протилежний

Лістинг програми LED3

```
// Interupt INT1 (D3)
// з'єднати D3 з будь-яким S1-S4

int led = 5;
volatile int state = LOW;

void setup(){
```

```
  pinMode(led, OUTPUT);
  attachInterrupt(1, blink, CHANGE);
  blink() ;
}

void loop(){
  digitalWrite(led, state);
}

void blink(){
  state = !state;

}
```

Програма LED4

Програма демонструє зміну яскравості світлодіода методом формування ШІМ сигналу. Алгоритм програми – подаємо на світлодіод 1/3 напруги від 0 до 5В (1,66В) з затримкою 250мс. Напрузі 5В відповідає код 255, напрузі 1,66В відповідає код (85) 255/3. Далі подаємо напругу 3,33В (код 170), що відповідає 2/3 яскравості світлодіода, та 5В (код 255), що відповідає максимальній яскравості.

Лістинг програми LED4

```
#define LED_PIN 6

void setup()
{
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  analogWrite(LED_PIN, 85);
  delay(250);

  analogWrite(LED_PIN, 170);
  delay(250);

  analogWrite(LED_PIN, 255);
  delay(250);
}
```

Програма LED5

Програма демонструє роботу з АЦП Arduino. Алгоритм програми – значення напруги з потенціометра RV1 зчитується внутрішнім АЦП і приймає значення в діапазоні від 0 до 1023. Це значення використовується для формування частоти мигання світлодіода HL1 та регулювання яскравості

світіння світлодіода HL6 (для цього отримане значення ділимо на 4 і отримаємо діапазон регулювання яскравості від 0 до 255).

Лістинг програми LED5

```
void setup() {
   pinMode(11, OUTPUT); //підключення світлодіода HL6
   pinMode(12, OUTPUT); // підключення світлодіода HL1
   pinMode(A1, INPUT);  // до входу A1 підключаємо потенціометр
}

void loop() {
   int val1 = analogRead(A1);
   int val2 = val1 / 4;
   analogWrite(11, val2);
   digitalWrite (12, HIGH);
   delay(val1) ;
   digitalWrite (12, LOW);
   delay(val1);
}
```

Додаток Г

Програма RGB1

Програма демонструє роботу з RGB світлодіодом. Алгоритм програми простий – по черзі включаємо червоний, зелений та синій колір RGB світлодіода з паузою в 0,5 секунди.

Лістинг програми RGB1

```
const byte rPin = 11;
const byte gPin = 10;
const byte bPin = 9;

void setup() {
    pinMode( rPin, OUTPUT );
    pinMode( gPin, OUTPUT );
    pinMode( bPin, OUTPUT );
}

void loop() {
    digitalWrite( bPin, LOW );//Blue-off
    digitalWrite( rPin, HIGH );//Red-on
    delay( 500 );

    digitalWrite( rPin, LOW ); //Red-off
    digitalWrite( gPin, HIGH ); //Green-on
    delay( 500 );

    digitalWrite( gPin, LOW ); //Green-off
    digitalWrite( bPin, HIGH );//Blue-on
    delay( 500 );
}
```

Програма RGB2

Програма демонструє роботу з RGB світлодіодом. Алгоритм програми простий – по черзі включаємо червоний, зелений та синій колір RGB світлодіода з паузою в 0,5 секунди з використанням масиву

Лістинг програми RGB2

```
const byte rgbPins[3] = {11,10,9};

void setup() {
    for( byte i=0; i<3; i++ )
        pinMode( rgbPins[i], OUTPUT );
}

void loop() {
    digitalWrite( rgbPins[2], LOW );
```

```
    digitalWrite( rgbPins[0], HIGH );
    delay( 500 );
    digitalWrite( rgbPins[0], LOW );
    digitalWrite( rgbPins[1], HIGH );
    delay( 500 );
    digitalWrite( rgbPins[1], LOW );
    digitalWrite( rgbPins[2], HIGH );
    delay( 500 );
}
```

Програма RGB3

Програма демонструє роботу з RGB світлодіодом. У програмі використовується масив для вибору R, G або B світлодіода та двомірний масив для вибору кольору світіння RGB світлодіода. Алгоритм програми простий – по черзі включається червоний, жовтий, зелений, блакитний, синій, фіолетовий колір RGB світлодіода з паузою в 1 секунду.

Лістинг програми RGB3

```
const byte rgbPins[3] = {11,10,9};
const byte rainbow[6][3] = {
    {1,0,0}, // red
    {1,1,0}, // yellow
    {0,1,0}, // green
    {0,1,1}, // light blue
    {0,0,1}, // blue
    {1,0,1}, // purple
};
void setup() {
    for( byte i=0; i<3; i++ )
        pinMode( rgbPins[i], OUTPUT );
}

void loop() {
     for( int i=0; i<6; i++ ){
         for( int k=0; k<3; k++ ){
            digitalWrite(rgbPins[k], rainbow[i][k]);
        }
        delay( 1000 );
    }
}
```

Програма RGB4

Програма демонструє роботу з RGB світлодіодом. У програмі використовується ШІМ для повільної зміни кольору світіння RGB світлодіоду. Алгоритм програми простий – повільно виключаємо один колір та одночасно повільно включаємо інший колір.

Лістинг програми RGB4

```
const byte rgbPins[3] = {11,10,9};
int dim = 1;

void setup() {
    for(byte i=0; i<3; i++){
        pinMode( rgbPins[i], OUTPUT );
    }
    // початковий стан  (Red - on)
    analogWrite(rgbPins[0], 255);
    analogWrite(rgbPins[1], 0);
    analogWrite(rgbPins[2], 0);
}

void loop() {
    for(int i=255; i>=0; i--){
        analogWrite( rgbPins[0], i/dim );//Red-off
        analogWrite( rgbPins[1], (255-i)/dim );//Green-on
        delay(10);
    }

    for(int i=255; i>=0; i--){
        analogWrite( rgbPins[1], i/dim );//Green-off
        analogWrite( rgbPins[2], (255-i)/dim );//Blue-on
        delay(10);
    }

    for(int i=255; i>=0; i--){
        analogWrite( rgbPins[2], i/dim );//Blue-off
        analogWrite( rgbPins[0], (255-i)/dim );//Red-on
        delay(10);
    }
}
```

Програма RGB5

Програма демонструє роботу з RGB світлодіодом. У програмі використовується лічильник натиснення тактової кнопки для зміни 7 кольорів світіння RGB світлодіода з використанням функції «антидребезгу». Алгоритм програми простий – у початковому стані RGB світлодіод виключений, з кожним натисненням кнопки змінюється 1 з 7 кольорів світіння світлодіоду.

Лістинг програми RGB5

```
#define BLED 9 //D9 - Blue
#define GLED 10 //D10 - Green
#define RLED 11 //D11 - Red
#define BUTTON 2 //D2 - button

boolean lastButton = LOW; //попередній стан кнопки
```

```
boolean currentButton = LOW; //поточний стан кнопки
int ledMode = 0; //режим роботи

void setup()
{
  pinMode (BLED, OUTPUT);
  pinMode (GLED, OUTPUT);
  pinMode (RLED, OUTPUT);
  pinMode (BUTTON, INPUT_PULLUP);
}

void loop()
{

  currentButton = debounce(lastButton); //читаємо стан кнопки
  if (lastButton == LOW && currentButton == HIGH) {
      ledMode++; }
  lastButton = currentButton;
  if (ledMode == 8) ledMode = 0;
  setMode(ledMode); //зміна режиму

//Функція антидребезгу
boolean debounce(boolean last)
{
  boolean current = digitalRead(BUTTON);
  if (last != current){
    delay(5);
    current = digitalRead(BUTTON);
  }
  return current;
}

//Вибір режиму роботи світлодіоду
void setMode(int mode)
{
  if (mode == 1)//Red
  {
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
  }
  else if (mode == 2)//Green
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
  }
  else if (mode == 3)//Blue
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, HIGH);
  }
```

```
    else if (mode == 4) // purple (Red + Blue)
  {
    analogWrite(RLED, 127);
    analogWrite(GLED, 0);
    analogWrite(BLED, 127);
  }
  else if (mode == 5) // turquoise (Blue + Green)
  {
    analogWrite(RLED, 0);
    analogWrite(GLED, 127);
    analogWrite(BLED, 127);
  }
  else if (mode == 6) //orange(Green + Red)
  {
    analogWrite(RLED, 127);
    analogWrite(GLED, 127);
    analogWrite(BLED, 0);
  }
  else if (mode == 7) //white(Red + Green + Blue)
  {
    analogWrite(RLED, 85);
    analogWrite(GLED, 85);
    analogWrite(BLED, 85);
  }
   else// off
  {
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);
  }
}
```

Додаток Д

Програма SEG1

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом. Алгоритм програми простий – на індикатор виводиться число 0123 в режимі динамічної індикації з частотою 100Гц (25 Гц на кожну позицію)

Лістинг програми SEG1

```
// Pin 2-8 is connected to the 7 segments of the display.
int pinA = 2;
int pinB = 3;
int pinC = 4;
int pinD = 5;
int pinE = 6;
int pinF = 7;
int pinG = 8;
int D1 = 9;
int D2 = 10;
int D3 = 11;
int D4 = 12;

void setup() {
// initialize the digital pins as outputs.
pinMode(pinA, OUTPUT);
pinMode(pinB, OUTPUT);
pinMode(pinC, OUTPUT);
pinMode(pinD, OUTPUT);
pinMode(pinE, OUTPUT);
pinMode(pinF, OUTPUT);
pinMode(pinG, OUTPUT);
pinMode(D1, OUTPUT);
pinMode(D2, OUTPUT);
pinMode(D3, OUTPUT);
pinMode(D4, OUTPUT);
}

void loop() {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, LOW);
  digitalWrite(D3, LOW);
  digitalWrite(D4, LOW);
  //0
  digitalWrite(pinA, HIGH);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, HIGH);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, HIGH);
```

```
  digitalWrite(pinG, LOW);
  delay(10);

  digitalWrite(D1, LOW);
  digitalWrite(D2, HIGH);
  digitalWrite(D3, LOW);
  digitalWrite(D4, LOW);
  //1
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, HIGH);
  digitalWrite(pinD, LOW);
  digitalWrite(pinE, LOW);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
  delay(10);

  digitalWrite(D1, LOW);
  digitalWrite(D2, LOW);
  digitalWrite(D3, HIGH);
  digitalWrite(D4, LOW);
  //2
  digitalWrite(pinA, HIGH);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, HIGH);
  delay(10);

  digitalWrite(D1, LOW);
  digitalWrite(D2, LOW);
  digitalWrite(D3, LOW);
  digitalWrite(D4, HIGH);
  //3
  digitalWrite(pinA, HIGH);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, HIGH);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, LOW);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, HIGH);
  delay(10);
/*
  //4
  digitalWrite(pinA, HIGH);
  digitalWrite(pinB, LOW);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
```

```
  delay(10);
  //5
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, LOW);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
  delay(10);
  //6
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, HIGH);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, LOW);
  digitalWrite(pinE, LOW);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
  delay(10);
  //7
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, LOW);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, HIGH);
  digitalWrite(pinG, HIGH);
  delay(10);
  //8
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, LOW);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, LOW);
  digitalWrite(pinE, LOW);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
  delay(10);
  //9
  digitalWrite(pinA, LOW);
  digitalWrite(pinB, LOW);
  digitalWrite(pinC, LOW);
  digitalWrite(pinD, HIGH);
  digitalWrite(pinE, HIGH);
  digitalWrite(pinF, LOW);
  digitalWrite(pinG, LOW);
  delay(10);
  */
}
```

Програма SEG2

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом та регістрами портів Arduino. Алгоритм програми простий – на індикатор виводиться число 0123 в режимі динамічної індикації з частотою 200Гц (50 Гц на кожну позицію)

Лістинг програми SEG2

```
/*
    A
   ---
F |   | B
  | G |
   ---
E |   | C
  |   |
   ---
    D
A-PD2, B-PD3, C-PD4, D- PD5, E-PD6, F-PD7
G-PB0, D1-PB1, D2-PB2, D3-PB3, D4-PB4 */
 # define P0 B00000010;
 # define P1 B00000100;
 # define P2 B00001000;
 # define P3 B00010000;

void setup() {
  DDRD=0xFF;
  DDRB=0xFF;
  PORTD =0x00;
  PORTB = 0<<0;
}
void loop() {
  PORTD=B11111100; //0
  PORTB=P0;
  delay (5);

  PORTD=B00011000;//1
  PORTB=P1;
  delay (5);

  PORTD=B01101100;//2
  PORTB=P2;
  PORTB|=1; //PB0=1
  delay (5);

  PORTD=B00111100;//3
  PORTB=P3;
  PORTB|=1; //PB0=1
  delay (5);
}
```

Програма SEG3

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом та регістрами портів Arduino. У програмі використовується оператор SWITCH та цикл FOR для вибору позиції та цифри та бітові операції bitSet, bitClear для установки / скидання сегменту G індикатора. Алгоритм програми простий – у кожну позицію індикатора по черзі виводиться цифра від 0 до 9 з затримкою 0,5 секунди.

Лістинг програми SEG3

```
//A-PD2, B-PD3, C-PD4, D-PD5, E-PD6, F-PD7
//G-PB0, D1-PB1, D2-PB2, D3-PB3, D4-PB4

 # define P0 B00000010;
 # define P1 B00000100;
 # define P2 B00001000;
 # define P3 B00010000;
 int tmp=0;

void setup() {
  DDRD=0xFF;
  DDRB=0xFF;
  PORTD =0x00;
  PORTB = 0<<0;
}

void loop() {
  for (int j=0; j<=3; j++){
    for (int i=0; i<10; i++){
      Cifra(i);
      delay(500);
    }
    switch (j){
      case 0:
        PORTB=P0;
        break;
      case 1:
        PORTB=P1;
        break;
      case 2:
        PORTB=P2;
        break;
      case 3:
        PORTB=P3;
        break;
      default:
        break;
       }
  }
}
```

```
void Cifra(int x){
  switch (x){
    case 0:
      PORTD=0xFC;
      break;
    case 1:
      PORTD=0x18;
      break;
    case 2:
      PORTD=0x6C;
      bitSet(PORTB,0);
      break;
    case 3:
      PORTD=0x3C;
      bitSet(PORTB,0);
      break;
   case 4:
      PORTD=0x98;
      bitSet(PORTB,0);
      PORTB|=1;
      break;
    case 5:
      PORTD=0xB4;
      bitSet(PORTB,0);
      break;
   case 6:
      PORTD=0xF4;
      bitSet(PORTB,0);
      break;
   case 7:
      PORTD=0x1C;
      bitClear(PORTB,0);
      break;
   case 8:
      PORTD=0xFC;
      bitSet(PORTB,0);
      break;
    case 9:
      PORTD=0xBC;
      bitSet(PORTB,0);
      break;
    default:
      break;
  }
}
```

Програма SEG4

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом та регістрами портів Arduino. У програмі використовується функція Cifra() та бітові операції bitSet, bitClear для формування цифри, яка буде виводитись на індикатор, у семисегментному коді. Алгоритм програми простий

– на індикатор виводиться число 5678 в режимі динамічної індикації з частотою 200Гц (50 Гц на кожну позицію).

Лістинг програми SEG4

```
//A-PD2, B-PD3, C-PD4, D- PD5, E-PD6, F-PD7
//G-PB0, D1-PB1, D2-PB2, D3-PB3, D4-PB4
# define P0 B00000010;
# define P1 B00000100;
# define P2 B00001000;
# define P3 B00010000;
int tmp=5;
int tmp1=6;
int tmp2=7;
int tmp3=8;

void setup() {
  DDRD=0xFF;
  DDRB=0xFF;
  PORTD =0x00;
  bitClear(PORTB,0);
}

void loop() {
 PORTB=P0;
 Cifra(tmp);
 delay (5);
 PORTB=P1;
 Cifra(tmp1);
 delay (5);
 PORTB=P2;
 Cifra(tmp2);
 delay (5);
 PORTB=P3;
 Cifra(tmp3);
 delay (5);
}

void Cifra(int x){
  switch (x)
  {
    case 0:
      PORTD=0xFC;
      break;
    case 1:
      PORTD=0x18;
      break;
    case 2:
      PORTD=0x6C;
      bitSet(PORTB,0);
      break;
    case 3:
```

```
        PORTD=0x3C;
        bitSet(PORTB,0);
        break;
    case 4:
        PORTD=0x98;
        bitSet(PORTB,0);
        //PORTB|=1;
        break;
     case 5:
        PORTD=0xB4;
        bitSet(PORTB,0);
        break;
    case 6:
        PORTD=0xF4;
        bitSet(PORTB,0);
        break;
    case 7:
        PORTD=0x1C;
        bitClear(PORTB,0);
        break;
    case 8:
        PORTD=0xFC;
        bitSet(PORTB,0);
        break;
     case 9:
        PORTD=0xBC;
        bitSet(PORTB,0);
        break;
     default:
        break;
  }
}
```

Програма SEG5

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом. У програмі використовується масиви *numeral[]*, *segmentPins[]*, *digitPins[]*, щоб задати конфігурацію підключення індикатора та представлення цифри, яка буде виводитись на індикатор, у семисегментному коді. У програмі використовується функції *showNumber()* та *showDigit()*, які організовують динамічну індикацію, переводять значення змінної *value* у BCD (двійково-десятковий) код, який відображається на індикаторі. Алгоритм програми простий – на індикатор виводиться число зі змінної *value* 5678 в режимі динамічної індикації з частотою 200Гц. Програму можна використовувати для відображення інформації з АЦП або аналогового датчика. Для цього потрібно їх значення зчитати командою *analogRead()* ло змінної *value*.

Лістинг програми SEG5

```
const int numeral[10] = {
  //ABCDEFGH
   B11111100, // 0
   B01100000, // 1
   B11011010, // 2
   B11110010, // 3
   B01100110, // 4
   B10110110, // 5
   B00111110, // 6
   B11100000, // 7
   B11111110, // 8
   B11100110, // 9
};

//                           H,G,F,E,D,C,B,A
const int segmentPins[] = {13,8,7,6,5,4,3,2};
const int nbrDigits= 4;

//digital                            0  1  2  3
const int digitPins[nbrDigits] = {9,10,11,12};

void setup()
{
   for(int i=0; i < 8; i++) {
      pinMode(segmentPins[i], OUTPUT);
   }
   for(int i=0; i < nbrDigits; i++) {
      pinMode(digitPins[i], OUTPUT);
   }
}

void loop()
{
   //int value = analogRead(0);
   int value = 1025;
   showNumber(value);
}

void showNumber( int number)
{
   if(number == 0) {
      showDigit( 0, nbrDigits-1) ;
   }
   else {
      for( int digit = nbrDigits-1; digit >= 0; digit--){
         if(number > 0){
            showDigit( number % 10, digit) ;
            number = number / 10;
         }
      }
   }
```

```
}

void showDigit( int number, int digit)
{
   digitalWrite( digitPins[digit], HIGH );
   for(int segment = 1; segment < 8; segment++)  {
      boolean isBitSet = bitRead(numeral[number], segment);
      digitalWrite( segmentPins[segment], isBitSet);
   }
   delay(5);
   digitalWrite( digitPins[digit], LOW );
}
```

Програма SEG6

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом. У програмі використовується переривання від Timer1 для формування динамічної індикації та переведення значення змінної *count* у BCD (двійково-десятковий) код. Функції *Disp()* виконує перекодування цифри у BCD коді в код семисегментного індикатора. Алгоритм програми простий – на індикатор виводиться число зі змінної *count* в режимі динамічної індикації. Значення змінної *count* збільшується з кожним натисненням тактової кнопки від 0 до 9999. Програму можна використовувати для більш складних проектів, наприклад, таймер, годинник, терморегулятор.

Лістинг програми SEG6

```
#define button A0

// segment pin definitions
#define SegA   2
#define SegB   3
#define SegC   4
#define SegD   5
#define SegE   6
#define SegF   7
#define SegG   8

// common pins of the four digits definitions
#define Dig1    9
#define Dig2    10
#define Dig3    11
#define Dig4    12

// variable declarations
byte current_digit;
int count = 0;

void setup()
```

```
{
  pinMode(button, INPUT_PULLUP);
  pinMode(SegA, OUTPUT);
  pinMode(SegB, OUTPUT);
  pinMode(SegC, OUTPUT);
  pinMode(SegD, OUTPUT);
  pinMode(SegE, OUTPUT);
  pinMode(SegF, OUTPUT);
  pinMode(SegG, OUTPUT);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
  pinMode(Dig3, OUTPUT);
  pinMode(Dig4, OUTPUT);

  disp_off();  // turn off the display

  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt
}

ISR(TIMER1_OVF_vect) // Timer1 interrupt service routine (ISR)
{
  disp_off();  // turn off the display

  switch (current_digit)
  {
    case 1:
      disp(count / 1000);
      digitalWrite(Dig1, HIGH);  // turn on digit 1
      break;

    case 2:
      disp( (count / 100) % 10);
      digitalWrite(Dig2, HIGH);     // turn on digit 2
      break;

    case 3:
      disp( (count / 10) % 10);
      digitalWrite(Dig3, HIGH);    // turn on digit 3
      break;

    case 4:
      disp(count % 10);
      digitalWrite(Dig4, HIGH);  // turn on digit 4
  }
   current_digit = (current_digit % 4) + 1;
}
// main loop
void loop()
{
```

```
  if(digitalRead(button) == 0)
  {
    count++;  // increment 'count' by 1
    if(count == 9999)
      count = 0;
    delay(200);  // wait 200 milliseconds
  }
}

void disp(byte number)
{
  switch (number)
  {
    case 0:  // print 0
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, LOW);
      break;

    case 1:  // print 1
      digitalWrite(SegA, LOW);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, LOW);
      break;

    case 2:  // print 2
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, LOW);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, HIGH);
      break;

    case 3:  // print 3
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, HIGH);
      break;
```

```
case 4:  // print 4
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 5:  // print 5
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 6:  // print 6
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 7:  // print 7
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 8:  // print 8
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 9:  // print 9
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
```

```
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
  }
}

void disp_off()
{
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
   digitalWrite(Dig4, LOW);
}

// end of code.
```

Додаток Е

Програма SHIFT

Програма демонструє роботу з 4 позиційним семисегментним індикатором з загальним катодом та регістром зсуву 74HC595. У програмі використовується переривання від Timer1 для формування динамічної індикації та переведення значення змінної *count* у BCD (двійково-десятковий) код. Функції *Disp()* виконує вивід цифри через регістр зсуву 74HC595 з використанням вбудованої функції *shiftOut()* у коді семисегментного індикатора. Алгоритм програми простий – на індикатор виводиться число зі змінної *count=9987* в режимі динамічної індикації. Значення змінної *count* збільшується з кожним натисненням тактової кнопки до 9999 із скиданням в 0. Програму можна використовувати для більш складних проектів, наприклад, таймер, годинник, терморегулятор для зменшення кількості (ущільнення) портів Arduino, що використовуються.

Лістинг програми SHIFT

```
//Q7-A, Q6-B,...Q0-H
//SC and RC - D7, SER-D6, SHIFT-VCC, OE-GND, Button-A0
#define button    A0
// shift register pin definitions
#define clockPin  7   // clock pin
#define dataPin   6   // data pin
// common pins of the four digits definitions
#define Dig1    5
#define Dig2    4
#define Dig3    3
#define Dig4    2

byte current_digit;
int  count = 9987;

void disp(byte number);

void setup()
{
  pinMode(button, INPUT_PULLUP);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
  pinMode(Dig3, OUTPUT);
  pinMode(Dig4, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  disp_off();  // turn off the display
  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
```

```
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt
}

ISR(TIMER1_OVF_vect) // Timer1 interrupt service routine (ISR)
{
  disp_off();  // turn off the display
  switch (current_digit)
  {
    case 1:
      disp(count / 1000); // prepare to display digit 1
      digitalWrite(Dig1, HIGH); // turn on digit 1
      break;

    case 2:
      disp( (count / 100) % 10 );//prepare to display digit 2
      digitalWrite(Dig2, HIGH);  // turn on digit 2
      break;

    case 3:
      disp( (count / 10) % 10 );// prepare to display digit 3
      digitalWrite(Dig3, HIGH); // turn on digit 3
      break;

    case 4:
      disp(count % 10);//prepare to display digit 4
      digitalWrite(Dig4, HIGH);  // turn on digit 4
  }

  current_digit = (current_digit % 4) + 1;
}

void loop(){
  if(digitalRead(button) == 0)
  {
    count++;  // increment 'count' by 1
    if(count > 9999)
      count = 0;
    delay(200);  // wait 200 milliseconds
  }
}

void disp(byte number){
  switch (number)
  {
    case 0:  // print 0
      shiftOut(dataPin, clockPin, MSBFIRST, 0xFC);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 1:  // print 1
      shiftOut(dataPin, clockPin, MSBFIRST, 0x60);
      digitalWrite(clockPin, HIGH);
```

```
      digitalWrite(clockPin, LOW);
      break;
    case 2:  // print 2
      shiftOut(dataPin, clockPin, MSBFIRST, 0xDA);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 3:  // print 3
      shiftOut(dataPin, clockPin, MSBFIRST, 0xF2);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 4:  // print 4
      shiftOut(dataPin, clockPin, MSBFIRST, 0x66);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 5:  // print 5
      shiftOut(dataPin, clockPin, MSBFIRST, 0xB6);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 6:  // print 6
      shiftOut(dataPin, clockPin, MSBFIRST, 0xBE);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 7:  // print 7
      shiftOut(dataPin, clockPin, MSBFIRST, 0xE0);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 8:  // print 8
      shiftOut(dataPin, clockPin, MSBFIRST, 0xFE);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
    case 9:  // print 9
      shiftOut(dataPin, clockPin, MSBFIRST, 0xF6);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
  }
}

void disp_off(){
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
   digitalWrite(Dig4, LOW);
}
```

Додаток Ж

Програма LCD1

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи .

Алгоритм програми простий – на індикатор (верхній рядок, 0 позиція) виводиться напис «VTC-2020». Далі курсор переводиться на нижній рядок, 0 позиція і виводиться кількість секунд з моменту запуску програми.

Лістинг програми LCD1

```
//Arduino pins 12, 11, 5, 4, 3, 2 to RS, E, D4, D5, D6, D7

#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("VTC-2020");
}

void loop() {
  // set the cursor to column 0, line 1
  lcd.setCursor(0, 1);

  // print the number of seconds since reset:
  lcd.print(millis() / 1000);

}
```

Програма LCD2

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи. Алгоритм програми простий – на індикаторі відображається рядок, що «біжить». Виводиться перший рядок, далі другий.

Лістинг програми LCD2

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(0, 0);
  // виводимо цифри від 0 до 9:
  for (int thisChar = 0; thisChar < 10; thisChar++) {
    lcd.print(thisChar);
    delay(500);
  }

  lcd.setCursor(16, 1);
  // включається автоматична прокрутка
  lcd.autoscroll();
  for (int thisChar = 0; thisChar < 10; thisChar++) {
    lcd.print(thisChar);
    delay(500);
  }
  // виключається автоматична прокрутка
  lcd.noAutoscroll();

  // очистка екрану
  lcd.clear();
}
```

Програма LCD3

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи. Алгоритм програми простий – виводимо рядок «hello, world!» та курсор, що мигає, у вигляді знакомісця.

Лістинг програми LCD3

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
}

void loop() {
  lcd.noBlink();
```

```
  delay(3000);
  lcd.blink();
  delay(3000);
}
```

Програма LCD4

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи. Алгоритм програми простий – виводимо рядок «hello, world!», що мигає

Лістинг програми LCD4

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
}
void loop() {
  lcd.noDisplay();
  delay(500);
  lcd.display();
  delay(500);
}
```

Програма LCD5

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи з символами користувача. Алгоритм програми простий – виводимо рядок «I♥Arduino☺» та анімацію чоловічка.

Лістинг програми LCD5

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte heart[8] = {
  0b00000,
  0b01010,
  0b11111,
  0b11111,
  0b11111,
  0b01110,
  0b00100,
  0b00000
};

byte smiley[8] = {
  0b00000,
```

```
  0b00000,
  0b01010,
  0b00000,
  0b00000,
  0b10001,
  0b01110,
  0b00000
};

byte frownie[8] = {
  0b00000,
  0b00000,
  0b01010,
  0b00000,
  0b00000,
  0b00000,
  0b01110,
  0b10001
};

byte armsDown[8] = {
  0b00100,
  0b01010,
  0b00100,
  0b00100,
  0b01110,
  0b10101,
  0b00100,
  0b01010
};

byte armsUp[8] = {
  0b00100,
  0b01010,
  0b00100,
  0b10101,
  0b01110,
  0b00100,
  0b00100,
  0b01010
};

void setup() {
  lcd.begin(16, 2);
  lcd.createChar(0, heart);
  lcd.createChar(1, smiley);
  lcd.createChar(2, frownie);
  lcd.createChar(3, armsDown);
  lcd.createChar(4, armsUp);
  lcd.setCursor(0, 0);

  lcd.print("I ");
  lcd.write(byte(0));
```

```
  lcd.print(" Arduino! ");
  lcd.write((byte)1);

}

void loop() {
  // read the potentiometer on A0:
  int sensorReading = analogRead(A0);
  // map the result to 200 - 1000:
  int delayTime = map(sensorReading, 0, 1023, 200, 1000);
  // set the cursor to the bottom row, 5th position:
  lcd.setCursor(4, 1);
  // draw the little man, arms down:
  lcd.write(3);
  delay(delayTime);
  lcd.setCursor(4, 1);
  // draw him arms up:
  lcd.write(4);
  delay(delayTime);
}
```

Програма LCD6

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі роботи. Алгоритм програми простий – виводимо рядок «hello, world!» який рухається праворуч та ліворуч

Лістинг програми LCD6

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
delay(1000);
}

void loop() {
  // зсув на 13 позицій (довжина рядка) ліворуч
  for (int position = 0; position < 13; position++) {
    // зсув на одну позицію
    lcd.scrollDisplayLeft();
    delay(150);
  }

  // зсув на 29 позицій праворуч (довж. рядка + довж. індик.)
  for (int position = 0; position < 29; position++) {
    // зсув на одну позицію праворуч
    lcd.scrollDisplayRight();
    delay(150);
  }
```

```
  // зсув на 16 позицій (довж. рядка + довж. індик) ліворуч
  // щоб повернути текст до центру
  for (int position = 0; position < 16; position++) {
    lcd.scrollDisplayLeft();
    delay(150);
  }

  delay(1000);

}
```

Програма LCD7

Програма демонструє роботу з LCD-індикатором 16×2 у 4 бітному режимі як індикатор прогресу яскравості світлодіоду. Алгоритм програми – зчитуємо значення з потенціометра та виводимо ці дані на екран у вигляді індикатора прогресу. Значення з потенціометра змінюються в діапазоні від 0 до 1024, тому їх необхідно перетворити в діапазон від 0 до 256 для світлодіода та від 0 до 17 для індикатора. Індикатор прогресу складається з закрашених прямокутників, які виводяться з початку рядка.

Лістинг програми LCD7

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int potPin = A0;       // потенціометр
int ledPin = 6;        // світлодіод на виводі з PWM
int potValue = 0;      // значення від потенціометра
int brightness = 0;    // яскравість
int progress = 0;      // індикатор прогресу

//progress bar character for brightness
byte pBar[8] = {
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
};

void setup()
{
  lcd.begin(16, 2);
  pinMode(ledPin, OUTPUT);
  lcd.print(" LED Brightness");
  lcd.createChar(0, pBar);
}
```

```
void loop()
{
   lcd.clear();
   lcd.print(" LED Brightness");
  lcd.setCursor(0, 1);

  potValue = analogRead(potPin);
  brightness = map(potValue, 0, 1024, 0, 255);
  analogWrite(ledPin, brightness);

  progress = map(brightness, 0, 255, 0, 17);
  for (int i = 0; i < progress; i++)
  {
    lcd.setCursor(i, 1);
    lcd.write(byte(0));
  }

  delay(750);
}
```

Додаток И

Програма LM35_SEG

Програма демонструє роботу з датчиком температури LM35 та 4 позиційним семисегментним індикатором з загальним катодом. У програмі використовується переривання від Timer1 для формування динамічної індикації. Функції Disp() виконує перекодування цифри в код семисегментного індикатора. Алгоритм програми – на індикатор виводиться значення температури в діапазоні від 0 до 99,9°C, яке нормовано до значення опорної напруги 1,1В, у режимі динамічної індикації. Значення температури відображається з одним знаком після коми.

Лістинг програми LM35_SEG

```
#define  LM35_pin  A0

// segment pin definitions
#define SegA 2
#define SegB 3
#define SegC 4
#define SegD 5
#define SegE 6
#define SegF 7
#define SegG 8
#define SegDP 13

// common pins of the four digits definitions
#define Dig1 9
#define Dig2 10
#define Dig3 11
#define Dig4 12

// variable declarations
byte current_digit;
int  temp;

void setup()
{
  pinMode(SegA, OUTPUT);
  pinMode(SegB, OUTPUT);
  pinMode(SegC, OUTPUT);
  pinMode(SegD, OUTPUT);
  pinMode(SegE, OUTPUT);
  pinMode(SegF, OUTPUT);
  pinMode(SegG, OUTPUT);
  pinMode(SegDP, OUTPUT);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
```

```
  pinMode(Dig3, OUTPUT);

  disp_off();  // turn off the display

  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt

  // set positive reference voltage to 1.1V
  analogReference(INTERNAL);
}

ISR(TIMER1_OVF_vect)
{
  disp_off();  // turn off the display

  switch (current_digit)
  {
    case 1:
        disp((temp / 100) % 10);
        digitalWrite(Dig1, HIGH);
        digitalWrite(SegDP, LOW);// turn on digit 1
      break;

    case 2:
      disp( (temp / 10) % 10);   // prepare to display digit 2
      digitalWrite(SegDP, HIGH);  // print decimal point ( . )
      digitalWrite(Dig2, HIGH);   // turn on digit 2
      break;

    case 3:
      disp(temp % 10);   // prepare to display digit 3
      digitalWrite(Dig3, HIGH);    // turn on digit 3
      digitalWrite(SegDP, LOW);
  }

  current_digit = (current_digit % 3) + 1;
}

void loop(){
  temp = 10* analogRead(LM35_pin)/9.355;
  // read analog voltage and convert it to B °C
  // (9.3 = 1023/(1.1*100))
  delay(1000);
}

void disp(byte number){
  switch (number)
  {
    case 0:  // print 0
      digitalWrite(SegA, HIGH);
```

```
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, LOW);
  break;

case 1:  // print 1
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 2:  // print 2
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, HIGH);
  break;

case 3:  // print 3
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, HIGH);
  break;

case 4:  // print 4
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 5:  // print 5
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
```

```
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 6:  // print 6
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, LOW);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 7:  // print 7
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, LOW);
      break;

    case 8:  // print 8
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 9:  // print 9
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
  }
}

void disp_off()
{
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
}
```

Програма LM35_Shift

Програма демонструє роботу з датчиком температури LM35, 4 позиційним семисегментним індикатором з загальним катодом та регістром зсуву 74HC595. У програмі використовується переривання від Timer1 для формування динамічної індикації. Функції Disp() виконує вивід цифри через регістр зсуву 74HC595 з використанням вбудованої функції shiftOut() у коді семисегментного індикатора. Алгоритм програми – на індикатор виводиться значення температури в діапазоні від 0 до 99,9°C, яке нормовано до значення опорної напруги 1,1В, у режимі динамічної індикації. Значення температури відображається з одним знаком після коми та символом «C».

Лістинг програми LM35_Shift

```
//7-segment display with 74HC595 shift register
//4-Digit counter example
//Common catode 7-segment display is used
//Q7-A, Q6-B,...Q0-H, SC and RC - D7, SER-D6, SHIFT-VCC, OE-GND

// counter button definition
#define button    A1
#define  LM35_pin A0

// shift register pin definitions
#define clockPin  7   // clock pin
#define dataPin   6   // data pin

// common pins of the four digits definitions
#define Dig1 5
#define Dig2 4
#define Dig3 3
#define Dig4 2

// variable declarations
byte current_digit;
int  temp;
void disp(int number, bool dec_point =0 );

void setup()
{
  pinMode(button, INPUT_PULLUP);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
```

```
  pinMode(Dig3, OUTPUT);
  pinMode(Dig4, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  disp_off();  // turn off the display

  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt

  //set positive reference voltage to 1.1V
  analogReference(INTERNAL);}

ISR(TIMER1_OVF_vect)   // Timer1 interrupt service routine (ISR)
{
  disp_off();  // turn off the display

  switch (current_digit)
  {
    case 1:
      disp( (temp / 100) % 10 );
      digitalWrite(Dig1, HIGH);  // turn on digit 1
      break;

    case 2:
      disp( (temp / 10) % 10, 1 );
      digitalWrite(Dig2, HIGH);     // turn on digit 2
      break;

    case 3:
      // prepare to display digit 3
      disp(temp % 10);
      digitalWrite(Dig3, HIGH);    // turn on digit 3
      break;

    case 4:
      disp(10);   // prepare to display digit 4 (most right)
      digitalWrite(Dig4, HIGH);  // turn on digit 4
      break;
  }

  current_digit = (current_digit % 4) + 1;
}
```

```
void loop(){
  temp = 10*analogRead(LM35_pin)/9.3;
  // read analog voltage and convert it to B°C
  // (9.3 = 1023/(1.1*100))
  delay(1000);  // wait 1 second
}

void disp(int number, bool dec_point){
  switch (number)
  {
    case 0:  // print 0
      shiftOut(dataPin, clockPin, MSBFIRST, 0xFC | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 1:  // print 1
      shiftOut(dataPin, clockPin, MSBFIRST, 0x60 | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 2:  // print 2
      shiftOut(dataPin, clockPin, MSBFIRST, 0xDA | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 3:  // print 3
      shiftOut(dataPin, clockPin, MSBFIRST, 0xF2 | dec_point);
      digitalWrite(clockPin, LOW);
      break;

    case 4:  // print 4
      shiftOut(dataPin, clockPin, MSBFIRST, 0x66 | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 5:  // print 5
      shiftOut(dataPin, clockPin, MSBFIRST, 0xB6 | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
```

```
    case 6:  // print 6
      shiftOut(dataPin, clockPin, MSBFIRST, 0xBE | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 7:  // print 7
      shiftOut(dataPin, clockPin, MSBFIRST, 0xE0 | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 8:  // print 8
      shiftOut(dataPin, clockPin, MSBFIRST, 0xFE | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 9:  // print 9
      shiftOut(dataPin, clockPin, MSBFIRST, 0xF6 | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;

    case 10:  // print C
      shiftOut(dataPin, clockPin, MSBFIRST, 0x9C | dec_point);
      digitalWrite(clockPin, HIGH);
      digitalWrite(clockPin, LOW);
      break;
  }
}

void disp_off(){
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
   digitalWrite(Dig4, LOW);
}
```

Програма LM35_LCD

Програма демонструє роботу з датчиком температури LM35 та LCD-індикатором. Алгоритм програми – на індикатор при старті виводиться напис «Digital Thermometer»; виконується вимірювання даних з LM35 та їх нормування до значення опорної напруги 1,1В. Значення температури відображається так: у

верхньому рядку виводиться по центру напис «Temperature»; у нижньому рядку значення температури з одним знаком після коми та символом «°C».

Лістинг програми LM35_LCD

```
// A0 - LM35
//Arduino pins 12, 11, 5, 4, 3, 2
//Arduino pins RS, E, D4, D5, D6, D7

#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

#define sensor A0
byte degree[8] =
{
0b00011,
0b00011,
0b00000,
0b00000,
0b00000,
0b00000,
0b00000,
0b00000
};
void setup(){
lcd.begin(16,2);
lcd.createChar(1, degree);
lcd.setCursor(0,0);
lcd.print(" Digital ");
lcd.setCursor(0,1);
lcd.print(" Thermometer ");
delay(2000);
lcd.clear();
// set positive reference voltage to 1.1V
analogReference(INTERNAL);
}

void loop(){
  /*---------Temperature-------*/
  float reading=analogRead(sensor);
  float temperature= reading /9.385;
  //read analog voltage and convert it to °C
  delay(10);

  /*------Display Result------*/
  lcd.clear();
  lcd.setCursor(2,0);
```

```
  lcd.print("Temperature");
  lcd.setCursor(4,1);
  lcd.print(temperature, 1);
  // 1 задає кількість знаків після коми
  lcd.write(1);
  lcd.print("C");

  delay(1000);
}
```

Додаток К

Програма DHT11_SEG

Програма демонструє роботу з датчиком температури DHT11 та 4 позиційним семисегментним індикатором з загальним катодом. У програмі використовується переривання від Timer1 для формування динамічної індикації. Функції Disp () виконує перекодування цифри в код семисегментного індикатора. Алгоритм програми – на індикатор виводиться значення температури в діапазоні від 0 до 99,9°С, у режимі динамічної індикації. Значення температури відображається з одним знаком після коми.

Лістинг програми DHT11_SEG

```
#include <DHT.h>
// define DHT data pin connection
#define DHTPIN 14 //Pin14--A0
DHT dht(DHTPIN, DHT11);

// segment pin definitions
#define SegA 2
#define SegB 3
#define SegC 4
#define SegD 5
#define SegE 6
#define SegF 7
#define SegG 8
#define SegDP 13

// common pins of the four digits definitions
#define Dig19
#define Dig210
#define Dig311
#define Dig4 12

// variable declarations
byte current_digit;
float t;
int temp;

void setup(){
  pinMode(SegA, OUTPUT);
  pinMode(SegB, OUTPUT);
  pinMode(SegC, OUTPUT);
  pinMode(SegD, OUTPUT);
```

```
  pinMode(SegE, OUTPUT);
  pinMode(SegF, OUTPUT);
  pinMode(SegG, OUTPUT);
  pinMode(SegDP, OUTPUT);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
  pinMode(Dig3, OUTPUT);

  disp_off();  // turn off the display
  dht.begin();

  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt
 }

ISR(TIMER1_OVF_vect) // Timer1 interrupt service routine (ISR)
{
  disp_off();  // turn off the display

  switch (current_digit)
  {
    case 1:
        disp(temp / 100);
        digitalWrite(Dig1, HIGH);
        digitalWrite(SegDP, LOW);// turn on digit 1
      break;

    case 2:
      disp((temp/10) % 10);   // prepare to display digit 2
      digitalWrite(SegDP, HIGH); // print point ( . )
      digitalWrite(Dig2, HIGH);   // turn on digit 2
      break;

    case 3:
      disp(temp % 10);   // prepare to display digit 3
      digitalWrite(Dig3, HIGH);    // turn on digit 3
      digitalWrite(SegDP, LOW);
  }

  current_digit = (current_digit % 3) + 1;
}

void loop(){
```

```
  t = dht.readTemperature();  // read  °C (
  temp=t*10;
  delay(1000);  // wait 1 second
}

void disp(int number){
  switch (number){
    case 0:  // print 0
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, LOW);
      break;

    case 1:  // print 1
      digitalWrite(SegA, LOW);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, LOW);
      break;

    case 2:  // print 2
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, LOW);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, HIGH);
      break;

    case 3:  // print 3
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, HIGH);
      break;
```

```
case 4:  // print 4
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 5:  // print 5
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 6:  // print 6
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 7:  // print 7
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 8:  // print 8
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
```

```
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 9:  // print 9
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
  }
}

void disp_off(){
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
}
```

Програма DHT11_LCD

Програма демонструє роботу з датчиком температури DHT11 та LCD-індикатором. Алгоритм програми – на індикаторі у верхньому рядку виводиться напис «Humidity = » та значення вологості; у нижньому рядку виводиться напис «Temp = » та значення значення температури.

Лістинг програми DHT11_LCD

```
// A0 - DHT11
//Arduino pins 12, 11, 5, 4, 3, 2
//Arduino pins RS, E, D4, D5, D6, D7

#include <LiquidCrystal.h>
#include "dht.h"

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int sensorPin = A0;
dht sensor;

void setup(){
  lcd.begin(16,2); //16 by 2 character display
  }
```

```
void loop(){
  delay(1000); //wait a sec (recommended for DHT11)
  sensor.read11(sensorPin);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Humidity = ");
  lcd.print(sensor.humidity);
  lcd.setCursor(0,1);
  lcd.print("Temp = ");
  lcd.print(sensor.temperature);
  delay(1000);
}
```

Лістинг програми DHT11_LCD_v2

```
// A0 - DHT11
//Arduino pins RS, E, D4, D5, D6, D7
//LiquidCrystal lcd(8,9,10,11,12,13);

#include <LiquidCrystal.h>
#include <DHT.h>
#define DHTPIN 14 //Pin14--A0
DHT dht(DHTPIN, DHT11);
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
float t;
float h;

void setup()
{
  lcd.begin(16,2); //16 by 2 character display
  dht.begin();
}

void loop()
{
  delay(1000); //wait a sec (recommended for DHT11)
  t = dht.readTemperature();
  h= dht.readHumidity();

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Humidity = ");
  lcd.print(h);
  lcd.setCursor(0,1);
  lcd.print("Temp = ");
  lcd.print(t);
}
```

Додаток Л

Програма DS1307_LCD

Програма демонструє роботу з годинником реального часу DS1307 та LCD-індикатором. У програмі використовується бібліотека DS1307.h, яка має бути встановлена або знаходитись у папці зі скетчем програми.. Алгоритм програми – на індикаторі у верхньому рядку виводиться напис «Date:» та дата; у нижньому рядку виводиться напис «Time:» та час. Кнопками S0–S3 можна встановити хвилини, години, день та місяць.

Лістинг програми DS1307_LCD

```
#include <LiquidCrystal.h>
#include "DS1307.h"
#include <Wire.h>
//Arduino pins RS, E, D4, D5, D6, D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte SW0 = A0; //SET MINUTES
byte SW1 = A1; //SET HOUR
byte SW2 = A2; //SET DAY
byte SW3 = A3; //SET MONTH
//byte SW4 = A6; //SET YEAR
int clock[7];

void setup(){

pinMode(SW0, INPUT);  // set minutes
pinMode(SW1, INPUT);  // set hours
pinMode(SW2, INPUT);  // set day
pinMode(SW3, INPUT);  // set month
//pinMode(SW4, INPUT);  // set year

digitalWrite(SW0, HIGH); // pull-ups on
digitalWrite(SW1, HIGH);
digitalWrite(SW2, HIGH);
digitalWrite(SW3, HIGH);
//digitalWrite(SW4,HIGH);

lcd.begin(20,2);
DS1307.begin();
//DS1307.setDate(20,8,22,6,10,52,04);
//рік, місяць, день, тиждень, години, хвилини, секунди
}

void loop(){
DS1307.getDate(clock);
```

```
lcd.setCursor(0,1);
lcd.print("Time: ");
Print(clock[4]);
lcd.print(":");
Print(clock[5]);
lcd.print(":");
Print(clock[6]);
lcd.setCursor(0,0);
lcd.print("Date: ");
Print(clock[1]);
lcd.print("/");
Print(clock[2]);
lcd.print("/");
lcd.print("20");
Print(clock[0]);

if(!digitalRead(SW0)){//minutes
  clock[5]++;
  if(clock[5]>59) clock[5]=0;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5], clock[6]);
  delay(100);
}

if(!digitalRead(SW1)){//hours
  clock[4]++;
  if(clock[4]>23) clock[4]=0;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5], clock[6]);
  delay(100);
}

if(!digitalRead(SW2)){//day
  clock[2]++;
  if(clock[2]>31) clock[2]=1;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5], clock[6]);
  delay (100);
}

if(!digitalRead(SW3)){//month
  clock[1]++;
  if(clock[1]>12) clock[1]=1;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5], clock[6]);
  delay (100);
}
/*
if(!digitalRead(SW4)){//year
  clock[0]++;
  if(clock[0]>99) clock[0]=0;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5], clock[6]);
```

```
  delay(100);
}
*/

delay(100);
}

// Функція для друку цифр 00,01,02,...
void Print(int number){
lcd.print(number/10);//друк старшого розряду
lcd.print(number%10);// друк молодшого розряду
}
```

Програма DS1307_SEG

Програма демонструє роботу з годинником реального часу DS1307 та семисегментним індикатором. У програмі використовується бібліотека DS1307.h, яка має бути встановлена або знаходитись у папці зі скетчем програми. У програмі використовується переривання від Timer1 для формування динамічної індикації. Функції Disp () виконує перекодування цифри в код семисегментного індикатора. Алгоритм програми – на індикатор виводиться час (години, хвилини). Кнопками S0–S2 можна встановити хвилини, години та скинути їх.

Лістинг програми DS1307_SEG

```
#include "DS1307.h"
#include <Wire.h>

// segment pin definitions
#define SegA   2
#define SegB   3
#define SegC   4
#define SegD   5
#define SegE   6
#define SegF   7
#define SegG   8
#define SegDP   13

// common pins of the four digits definitions
#define Dig1    9
#define Dig2    10
#define Dig3    11
#define Dig4    12

// variable declarations
```

```
byte current_digit;
byte SW0 = A0; //RESET
byte SW1 = A1; //SET MINUTES
byte SW2 = A2; //SET HOUR

int clock[7];

void setup() {

  pinMode(SegA, OUTPUT);
  pinMode(SegB, OUTPUT);
  pinMode(SegC, OUTPUT);
  pinMode(SegD, OUTPUT);
  pinMode(SegE, OUTPUT);
  pinMode(SegF, OUTPUT);
  pinMode(SegG, OUTPUT);
  pinMode(SegDP, OUTPUT);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
  pinMode(Dig3, OUTPUT);
  pinMode(Dig4, OUTPUT);

  pinMode(SW0, INPUT);
  pinMode(SW1, INPUT);
  pinMode(SW2, INPUT);
  digitalWrite(SW0, HIGH); // pull-ups on
  digitalWrite(SW1, HIGH);
  digitalWrite(SW2, HIGH);

  DS1307.begin();
  //DS1307.setDate(20,7,18,0,17,36,54);//Y,M,D,W,H,M,S

  disp_off();  // turn off the display
  DS1307.getDate(clock);
}

void loop() {
DS1307.getDate(clock);
Indicate();


if(!digitalRead(SW2)){
  clock[4]++;
  if(clock[4]>23) clock[4]=0;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5],clock[6]);
```

```
  delay(20);
}

if(!digitalRead(SW1)){
  clock[5]++;
  if(clock[5]>59) clock[5]=0;
  DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
  clock[5],clock[6]);
  delay(20);
}

if(!digitalRead(SW0)){ // clear time
   clock[5]=0;
   clock[4]=0;
   DS1307.setDate(clock[0],clock[1],clock[2],0,clock[4],
   clock[5],clock[6]);
   delay(20);
 }
 delay (100);
}

void disp(int number){
  switch (number)
  {
    case 0:  // print 0
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, LOW);
      break;

    case 1:  // print 1
      digitalWrite(SegA, LOW);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, LOW);
      break;

    case 2:  // print 2
      digitalWrite(SegA, HIGH);
```

```
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, HIGH);
  break;

case 3:  // print 3
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, HIGH);
  break;

case 4:  // print 4
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 5:  // print 5
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;

case 6:  // print 6
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, HIGH);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
```

```
      break;

    case 7:  // print 7
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, LOW);
      break;

    case 8:  // print 8
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 9:  // print 9
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, HIGH);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
  }
}

void disp_off(){
   digitalWrite(Dig1, LOW);
   digitalWrite(Dig2, LOW);
   digitalWrite(Dig3, LOW);
   digitalWrite(Dig4, LOW);
}

void Indicate(){
  disp_off();  // turn off the display
  switch (current_digit)
  {
    case 1:
       disp(clock[4]/10);
```

```
        digitalWrite(Dig1, HIGH);
        digitalWrite(SegDP, LOW);// turn on digit 1
        break;

    case 2:
      disp(clock[4]%10);   // prepare to display digit 2
      digitalWrite(SegDP, HIGH);  // print decimal point ( . )
      digitalWrite(Dig2, HIGH);   // turn on digit 2
      break;

    case 3:
      disp(clock[5]/10);   // prepare to display digit 3
      digitalWrite(Dig3, HIGH);    // turn on digit 3
      digitalWrite(SegDP, LOW);
      break;

    case 4:
      disp(clock[5]%10);   // prepare to display digit 3
      digitalWrite(Dig4, HIGH);    // turn on digit 4
      digitalWrite(SegDP, LOW);

  }

  current_digit = (current_digit % 4) + 1;

}
```

Додаток М

Програма Matrix_One

Програма демонструє роботу з матричним дисплеєм 8×8 та регістром керування MAX7219 з інтерфейсом SPI. У програмі використовуються бібліотекі LedControl.h, binary.h, які мають бути встановлена або знаходитись у папці зі скетчем програми. Алгоритм програми – на матричному дисплеї 8×8 виводиться зображення цифри 1.

Лістинг програми Matrix_One

```
#include "LedControl.h"
#include "binary.h"
/*
 DIN connects to pin 12
 CLK connects to pin 11
 CS connects to pin 10
*/
LedControl lc=LedControl(12,11,10,1);
unsigned long delaytime=1000;

//one
byte one[8]= {
B00001100,
B00010100,
B00100100,
B00000100,
B00000100,
B00000100,
B00000100,
B11111111
};

void setup() {
  lc.shutdown(0,false);
  // Set brightness to a medium value
  lc.setIntensity(0,8);
  // Clear the display
  lc.clearDisplay(0);
}

void drawDigital(){
  // Display one
  lc.setRow(0,0,one[0]);
```

```
  lc.setRow(0,1,one[1]);
  lc.setRow(0,2,one[2]);
  lc.setRow(0,3,one[3]);
  lc.setRow(0,4,one[4]);
  lc.setRow(0,5,one[5]);
  lc.setRow(0,6,one[6]);
  lc.setRow(0,7,one[7]);
  delay(delaytime);
}

void loop(){
  drawDigital();
}
```

Програма Matrix_Smile

Програма демонструє роботу з матричним дисплеєм 8×8 та регістром керування MAX7219 з інтерфейсом SPI. У програмі використовуються бібліотекі LedControl.h, binary.h, які мають бути встановлена або знаходитись у папці зі скетчем програми. Алгоритм програми – на матричному дисплеї 8×8 виводиться по черзі смайліки (sad face, neutral face, happy face) у виглядф анімації.

Лістинг програми Matrix_ Smile

```
#include "LedControl.h"
#include "binary.h"
/*
 DIN connects to pin 12
 CLK connects to pin 11
 CS connects to pin 10 */
LedControl lc=LedControl(12,11,10,1);
unsigned long delaytime=1000;

// happy face
byte hf[8]= {
B00111100,B01000010,B10100101,B10000001,
B10100101,B10011001,B01000010,B00111100};
// neutral face
byte nf[8]={
B00111100,B01000010,B10100101,B10000001,
B10111101,B10000001,B01000010,B00111100};
// sad face
byte sf[8]= {
B00111100,B01000010,B10100101,B10000001,
B10011001,B10100101,B01000010,B00111100};
```

```
void setup() {
  lc.shutdown(0,false);
  // Set brightness to a medium value
  lc.setIntensity(0,8);
  // Clear the display
  lc.clearDisplay(0);  }

void drawFaces(){
  // Display sad face
  lc.setRow(0,0,sf[0]);
  lc.setRow(0,1,sf[1]);
  lc.setRow(0,2,sf[2]);
  lc.setRow(0,3,sf[3]);
  lc.setRow(0,4,sf[4]);
  lc.setRow(0,5,sf[5]);
  lc.setRow(0,6,sf[6]);
  lc.setRow(0,7,sf[7]);
  delay(delaytime);
  // Display neutral face
  lc.setRow(0,0,nf[0]);
  lc.setRow(0,1,nf[1]);
  lc.setRow(0,2,nf[2]);
  lc.setRow(0,3,nf[3]);
  lc.setRow(0,4,nf[4]);
  lc.setRow(0,5,nf[5]);
  lc.setRow(0,6,nf[6]);
  lc.setRow(0,7,nf[7]);
  delay(delaytime);
  // Display happy face
  lc.setRow(0,0,hf[0]);
  lc.setRow(0,1,hf[1]);
  lc.setRow(0,2,hf[2]);
  lc.setRow(0,3,hf[3]);
  lc.setRow(0,4,hf[4]);
  lc.setRow(0,5,hf[5]);
  lc.setRow(0,6,hf[6]);
  lc.setRow(0,7,hf[7]);
  delay(delaytime);
}
void loop(){
  drawFaces();}
```

Навчальне видання

Денисюк Валерій Олександрович

Цирульник Сергій Михайлович

МІКРОПРОЦЕСОРНІ СИСТЕМИ УПРАВЛІННЯ

Навчальний посібник

Набір і редагування авторські

Технічний редактор: *Цирульник Сергій Михайлович*