В.О. Денисюк, С.М. Цирульник

МІКРОПРОЦЕСОРНІ СИСТЕМИ УПРАВЛІННЯ

Навчальний посібник

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)
М59

Рекомендовано Вченою радою як навчальний посібник для студентів галузі знань 12 «Інформаційні технології» (протокол № 9 від 23.03.2021р.)

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.

Денисюк В.О., Цирульник С.М.

М59 Мікропроцесорні системи управління: навч. посіб./ В.О.Денисюк, С.М.Цирульник; Вінн. нац. аграр. ун-т. Вінниця: ТВОРИ, 2021. 204 с.

ISBN 978-966-949-833-5

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 978-966-949-833-5

© В.О. Денисюк, С. М. Цирульник, 2021
© ВНАУ, 2021

ЗМІСТ

ВСТУП

Метою вивчення дисципліни «Мікропроцесорні системи управління» є засвоєння студентами методів розробки мікропроцесорних пристроїв управління на базі мікроконтролерів. Основними завданнями навчальної дисципліни «Мікропроцесорні системи управління» є вивчення апаратної будови мікропроцесорних систем управління й основ їх програмування.

У результаті вивчення навчальної дисципліни студент повинен знати, вміти та мати такі навички.

Знати: структури, принцип дії і методики програмування мікропроцесорних систем управління, які будуються на базі мікроконтролерів; особливості застосування мікроконтролерів для побудови електронних систем управління;

Вміти: застосовувати мікроконтролери для побудови мікропроцесорних систем управління; оцінювати техніко-економічну ефективність схемотехнічних рішень, які використовуються при розробці мікропроцесорних систем управління; програмувати найбільш поширені мікроконтролери; налагоджувати і діагностувати мікропроцесорні системи управління;

Мати навички та компетентності: спілкування, включаючи усну та письмову комунікацію українською мовою та однією з іноземних мов; використання різноманітних методів, зокрема інформаційних технологій, для ефективно спілкування на професійному та соціальному рівнях; здатність адаптуватись до нових ситуацій та приймати рішення; відповідальне ставлення до виконуваної роботи та досягати поставленої мети з дотриманням вимог професійної етики.

Метою практичних занять з дисципліни «Мікропроцесорні системи управління», виконуваних за допомогою програмного забезпечення в середовищі програмування мікропроцесорів Arduino NANO, лабораторного макету та/або віртуальної моделі лабораторного макету з використанням програмного та інформаційного забезпечення засобів Proteus 8 VSM, Arduino

Uno, ArduSim V2.6, є: вивчення принципів побудови електронних компонентів мікропроцесорних систем на основі процесорів Arduino NANO; придбання навичок роботи з сенсорами та датчиками з мікропроцесорним управлінням; придбання навичок підготовки, проведення та документування результатів вимірювання і функціонування мікропроцесорних систем.

У результаті виконання практичних робіт студент повинен вміти формулювати вимоги до параметрів мікропроцесорної системи управління. Для проектування мікропроцесорної системи управління необхідно навчитись враховувати такі основні характеристики: можливості мікропроцесорної системи; обсяг пам’яті програм та даних мікропроцесорної системи управління; набір команд та способів адресації; розрядність та швидкодія; вимоги до джерела живлення та споживаної потужності; вартість мікропроцесорної системи управління в різних варіантах виконання; ефективність засобів програмування та налагодження мікропроцесорних систем управління з використанням програмного та інформаційного забезпечення Proteus 8 VSM, Arduino UnoArduSim V2.6.

Завдання розраховані на 2 академічні години. Результати проведених досліджень повинні бути задокументовані та в кінці заняття представлені викладачу. До виконаної практичної роботи складається звіт, який повинен містити: титульний аркуш звіту; теоретичні відомості з теми дослідження; схеми, на яких проводилися дослідження та налагодження мікропроцесорної системи управління; програмний код, що ілюструє досліджені процеси; висновки за результатами досліджень та вимірів; відповіді на контрольні питання.

РОЗДІЛ 1. ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ УПРАВЛІННЯ

1.1 Основні поняття мікропроцесорних систем

Мікропроцесор (МП), мікроконтролер (МК), мікрокомп'ютер (МКП) слова схожі, але за змістом різні. Спрощена структурна схема типового мікропроцесора показана на рис 1.1. У його основі. центральний процесорний пристрій (ЦПП), який містить арифметичний обчислювач, логічне ядро і регістри загального призначення. Із зовнішнім світом ЦПП спілкується за допомогою трьох шин: адреси, даних і управління. По цих же шинах в нього поступають коди програми керування, яка зберігається на зовнішньому носії. Початкова установка регістрів ЦПП виконується по сигналу скидання RESET, а синхронізація роботи здійснюється від тактових імпульсів SYN.

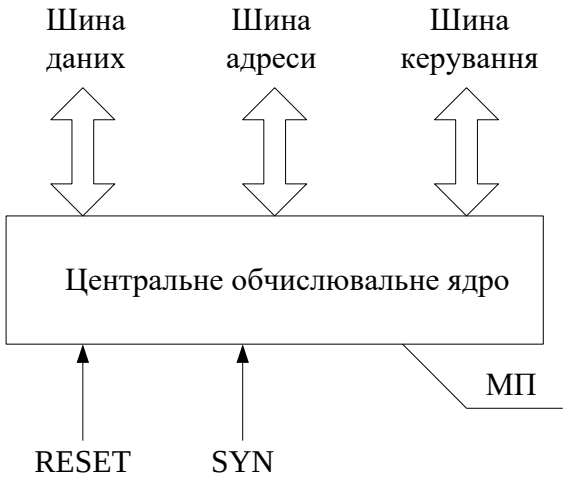


Рис.1.1. Функціональна схема мікропроцесора (МП)

Якщо до ЦПП на кристал додати оперативний і постійний запам’ятовуючий пристрій (ОЗП, ПЗП), таймери, лічильники, аналого-цифрові і цифро-аналогові перетворювачі (АЦП, ЦАП), інтерфейсні вузли і порти введення/виведення, то мікропроцесор перетвориться на МК (рис.1.2). Тактові імпульси виробляє вбудований синхрогенератор, частота якого стабілізується кварцовим резонатором. Для програмування ПЗП використовується окремий

вхід PROG або шина з декількох сигналів.

Раніше МК називали однокристальними мікро ЕОМ, віддаючи належне вітчизняним мікросхемам К1816ВЕхх, КР1830ВЕхх. Однак ця назва не закріпилася, оскільки зараз переважні позиції на ринку займають МК (microcontroller) закордонного виробництва сімейств AVR, MCS-51, PIC, Scenix, Z8.

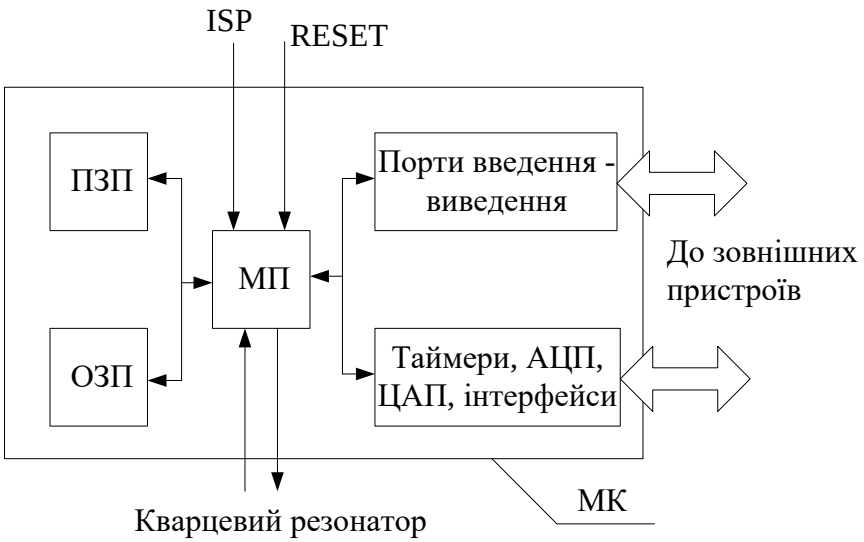


Рис.1.2. Функціональна схема мікроконтролера (МК)

Мікроконвертор. це вдалий рекламний винахід фірми Analog Devices. Першим мікроконвертором був ADUC812, випущений в 1998 р. Ключове слово «MicroConverter» є офіційною торгівельною маркою і захищено юридичними правами фірми Analog Devices. Відноситься воно до лінійок мікросхем ADuC7xxx, ADuC8xx, що виконують функцію центрального ядра інтелектуальних систем збору інформації.

«Родзинкою» мікроконверторів є швидкодіючий прецизійний АЦП, доповнений універсальним блоком логічної обробки даних і багато розрядним ЦАП. Якщо врахувати наднизьке споживання струму і малі габарити мікроконверторів, то стає ясно, що спеціалізовані ІМС по праву займають свою нішу на ринку.

Проте, структурні схеми в мікроконверторів і МК повністю збігаються.

Проте принципова різниця все ж є. Для звичайного МК спочатку вибирається цифрове обчислювальне ядро, а потім до нього додається АЦП і ЦАП. В протилежність цьому, ядром мікроконвертора спочатку служить зв'язка прецизійних АЦП і ЦАП, до яких додається процесор, що управляє.

Цифрові сигнальні процесори (англ. DSP - Digital Signal Processor) теж відносяться до мікроконтролерних пристроїв (рис.1.3). Їх особливістю є обробка широкосмугових сигналів в режимі реального часу. Це характерно як для аудіо/відео техніки, так і для систем гнучкого управління роботизованими комплексами. Досягненню мети сприяє висока швидкодія ядра сигнального процесора (СП), багатопотокова система обслуговування пам'яті і наявність апаратних математичних команд, наприклад, для швидкого перетворення Фур'є. Звичайні МК такими можливостями не володіють.

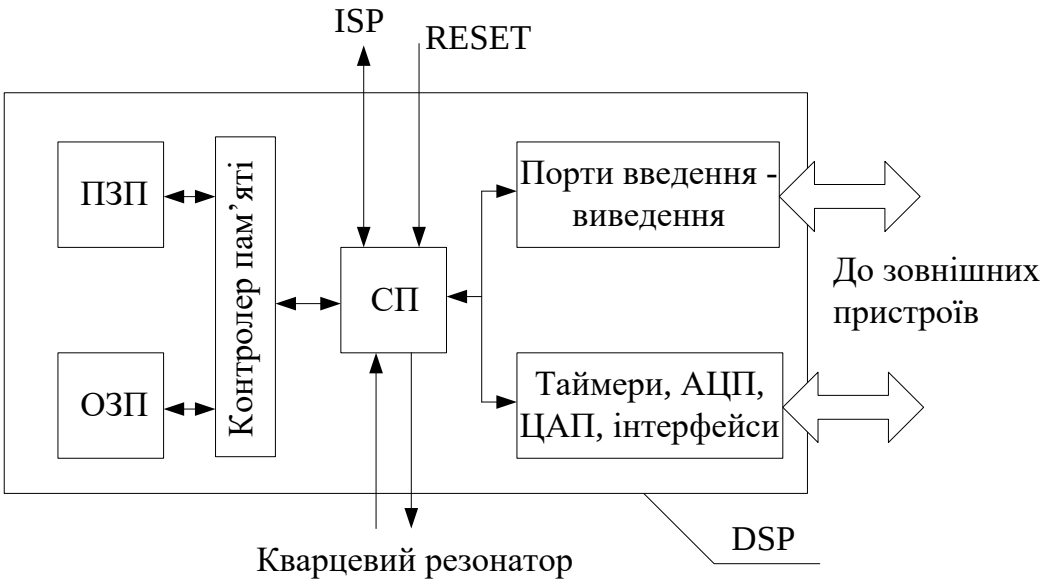


Рис.1.3. Функціональна схема DSP

Перші DSP з'явилися в кінці 1970-х років через декілька років після перших МК, проте висока ціна і технологічні обмеження того часу не дозволили їм змагатися на рівних. Лише у 1982 р. фірма Texas Instruments зробила революційний прорив, випустивши в продаж перший універсальний програмований DSP TMS32010. Його концепція стала стандартом «де-факто»

для всіх подальших сигнальних процесорів.

Відмінності в архітектурі і вузька спеціалізація привели до того, що напрям DSP/DSC виділився в окрему від МК сферу розробок з кількістю різновидів моделей більше 300. Вважається, що основною відмінністю DSP є відсутність розвиненої системи команд управління, тобто умовних переходів, непрямих викликів підпрограм, які необхідні для виконання завдань з'єднання із зовнішніми об'єктами. Процесор в DSP і його системі команд орієнтовані на найвищу швидкість перетворення вхідних даних, що поступають. На управлінські «дрібниці» обчислювальних ресурсів вже не вистачає.

Сучасні МК запозичують від DSP апаратне множення і спеціалізацію команд, а DSP запозичують від МК універсальні інтерфейси введення/виведення і гнучкість в платформі програмування. Грані відмінностей поступово стираються.

На початку 1980-х років японська фірма Hitachi почала використовувати термін «мікрокомп'ютер», яким стали називати швидкодіючі процесори лінійки «Hitachi SUPERH». У рекламі можливостей чипів «SUPERH microcomputer SH7000 series» підкреслювалося, що на одній мікросхемі тепер можна побудувати систему керування реального часу, що перевищує за продуктивністю звичайний настільний мікрокомп'ютер.

Сучасний мікрокомп'ютер (рис.1.4) містить всі складові МК або DSP, але додатково має контролер шин для підключення зовнішньої високошвидкісної пам'яті, а також аудіо- і відеопроцесори. Останні, як правило, не поступаються ЦПП за складністю та функціональністю. Приклади спрощених мікрокомп'ютерів з повсякденного життя. це однокристальні ВІС китайських клонів ігрових приставок «Dendy», «SEGA Mega Drive».

Мікропроцесор. це пристрій, який здійснює прийом, обробку і видачу інформації. Конструктивно МП містить одну або декілька інтегральних схем і виконує дії за програмою, записаною в пам'яті.

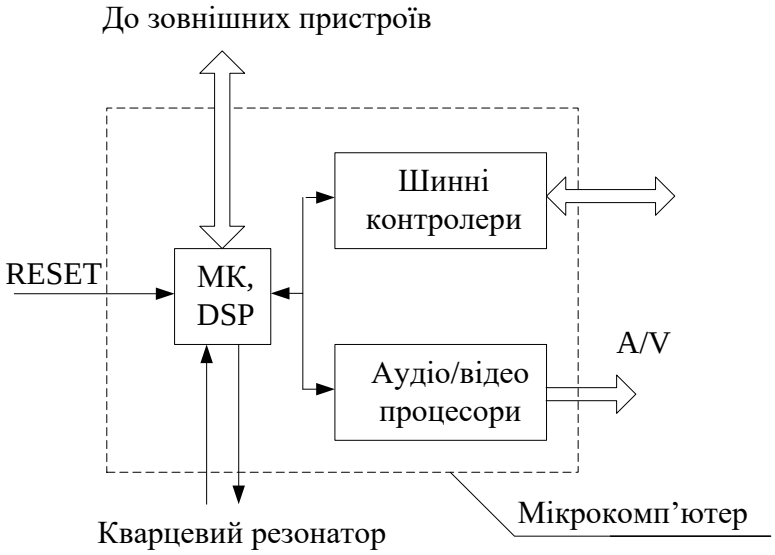


Рис.1.4 . Функціональна схема мікрокоп‘ютера

Мікропроцесорна система. обчислювальна, контрольно-вимірювальна або система керування, в якій основним пристроєм обробки інформації є МП. Мікропроцесорна система будується з набору мікропроцесорних ВІС.

Мультимікропроцесорна (або мультипроцесорна) система. система, яка утворюється об'єднанням деякої кількості універсальних або спеціалізованих МП, завдяки чому забезпечується паралельна обробка інформації і розподілене керування.

Мікропроцесорний комплект (МПК). сукупність інтегральних схем, сумісних за електричними, інформаційними та конструктивними параметрами і призначених для побудови мікропроцесорних систем керування. Зазвичай МПК містить: ВІС МП (один чи кілька корпусів інтегральних схем); ВІС оперативних запам'ятовувальних пристроїв (ОЗП); ВІС постійних запам'ятовувальних пристроїв (ПЗП); інтерфейси або контролери зовнішніх пристроїв; службові ВІС (тактовий генератор, регістри, шинні формувачі, контролери шин, арбітри шин).

Жорстке розділення мікросхем на МК, мікропроцесори і DSP було характерне в кінці XX століття. У сьогодення грані відмінностей поступово стираються. МК усе частіше відносять до класу процесорів для вбудованих

застосувань або, по-іншому, процесорів вбудованих систем (embedded processor). За визначенням, вбудовані обчислювальні системи. це системи, які безпосередньо, без постійної присутності людини, взаємодіють з датчиками і виконавчими пристроями керованого об'єкту.

Прикладами вбудованих систем є бортові і панельні комп'ютери, портативні вимірювальні прилади, системи відеоспостереження, роботи, мережеве устаткування, стільникові телефони.

Для досягнення максимальної універсальності і спрощення протоколів обміну інформацією в мікропроцесорних системах застосовується шинна структура зв'язку між окремими пристроями, що входять в систему.

При шинній структурі зв'язку (рис.1.5) всі сигнали між пристроями передаються по одних і тих же лініях зв'язку, але у різний час. Передача по всіх лініях зв'язку може здійснюватися в обох напрямах. У результаті кількість ліній зв'язку істотно скорочується, а правила обміну (протоколи) спрощуються. Група ліній зв'язку, по яким передаються сигнали або коди називається шиною (англ. bus). При шинній структурі зв'язку легко здійснюється пересилка всіх інформаційних потоків у потрібному напрямку, наприклад, їх можна пропустити через один процесор, що дуже важливо для мікропроцесорної системи. Проте при шинній структурі зв'язку вся інформація передається по лініях зв'язку послідовно в часі, по черзі, що знижує швидкодію системи у порівнянні з класичною структурою зв'язку.

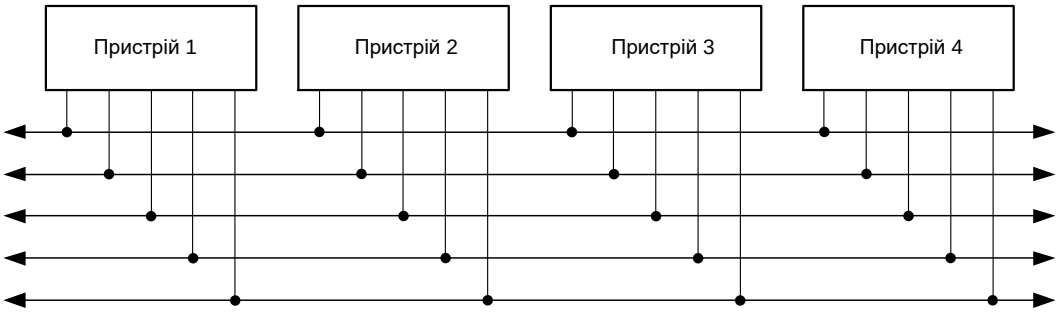


Рис. 1.5 . Шинна структура зв’язків

Перевагою шинної структури зв'язку є те, що всі пристрої, що підключені до шини, повинні приймати і передавати інформацію за одними і тими ж

правилами (протоколам обміну інформацією по шині). Відповідно, всі вузли, що відповідають за обмін з шиною в цих пристроях, повинні бути одноманітні, уніфіковані.

Недоліком шинної структури є те, що всі пристрої підключаються до кожної лінії зв'язку паралельно. Тому будь-яка несправність будь-якого пристрою може вивести з ладу всю систему. З цієї ж причини настройка системи з шинною структурою зв'язку досить складна і вимагає спеціального устаткування.

Типова структура мікропроцесорної системи наведена на рис.1.6. Вона включає три основних типу пристроїв: процесор; пам'ять, що включає оперативну пам'ять ОЗП і постійну пам'ять ПЗП, яка служить для зберігання даних і програм; пристрої введення/виведення (ПВВ, I/O - Input/Output Devices), які призначені для зв’язку мікропроцесорної системи із зовнішніми пристроями; для приймання (введення, читання, Read) вхідних сигналів і передавання (виведення, запис, Write) вихідних сигналів.

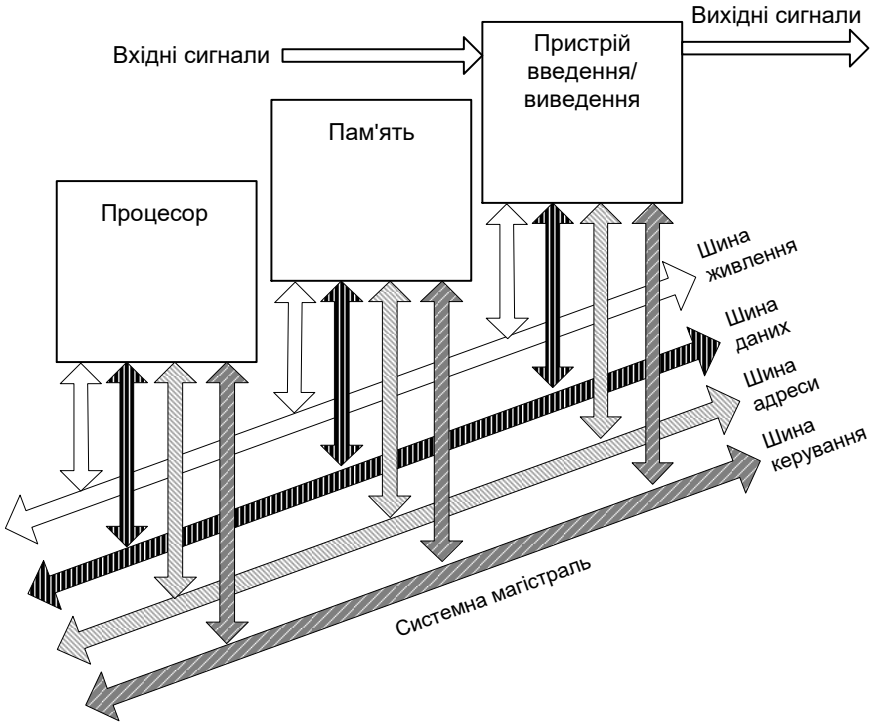


Рис.1.6 . Структура мікропроцесорної системи

Усі пристрої мікропроцесорної системи об'єднуються загальною

системною шиною (вона ж називається ще системною магістраллю або каналом). Системна магістраль включає чотири основні шини нижнього рівня: шина адреси (Address Bus); шина даних (Data Bus); шина керування (Control Bus); шина живлення (Power Bus).

Шина адреси служить для визначення адреси пристрою, з яким процесор обмінюється інформацією в даний момент. Кожному пристрою (окрім процесора), кожному елементу пам'яті в мікропроцесорній системі задається власна адреса. Коли код якоїсь адреси виставляється процесором на шині адреси, пристрій, якому ця адреса призначена, розуміє, що його чекає обмін інформацією.

Шина даних. це основна шина, яка використовується для передачі інформаційних кодів між всіма пристроями мікропроцесорної системи. Звичайно в пересилці інформації бере участь процесор, який передає код даних в якийсь пристрій або в елемент пам'яті чи ж приймає код даних з якогось пристрою або з елементу пам'яті. Але можлива також і передача інформації між пристроями без участі процесора. Шина даних завжди двохнаправлена.

Шина керування на відміну від шини адреси і шини даних складається з окремих сигналів керування. Кожний з цих сигналів під час обміну інформацією має свою функцію. Деякі сигнали служать для того щоб синхронізувати дані, що передаються або приймаються. Інші сигнали керування можуть використовуватися для підтвердження прийому даних, для скидання всіх пристроїв в початковий стан. Лінії шини керування можуть бути однонаправленими або двохнаправленими.

Шина живлення призначена не для пересилки інформаційних сигналів, а для живлення системи. Вона складається з ліній живлення і загального дроту. У мікропроцесорній системі може бути одне джерело живлення (частіше +5В) або декілька джерел живлення (звичайно ще -5В +12В і -12В). Кожній напрузі живлення відповідає своя лінія зв'язку. Всі пристрої підключені до цих ліній паралельно.

У мікропроцесорній системі всі інформаційні коди і коди команд передаються по шинах послідовно, по черзі. Це визначає порівняно невисоку швидкодію мікропроцесорної системи. Воно обмежене навіть не швидкодією процесора (яке теж дуже важливо) і не швидкістю обміну по системній шині, а саме послідовним характером передачі інформації по системній шині.

1.2 Архітектура AVR мікроконтролерів

На рис.1.7 наведена узагальнена структурна схема плати Arduino-UNO. Ядром Arduino є AVR-контролер, що тактується від кварцового резонатора частотою 16 МГц. Лінії портів МК виводяться назовні на контактну «гребінку» плати без яких-небудь обмежувальних або захисних елементів. Початкове скидання проводиться кнопкою SB1. На платі є 4 світлодіодних індикатора, з яких 3 службові та один («L») користувача.

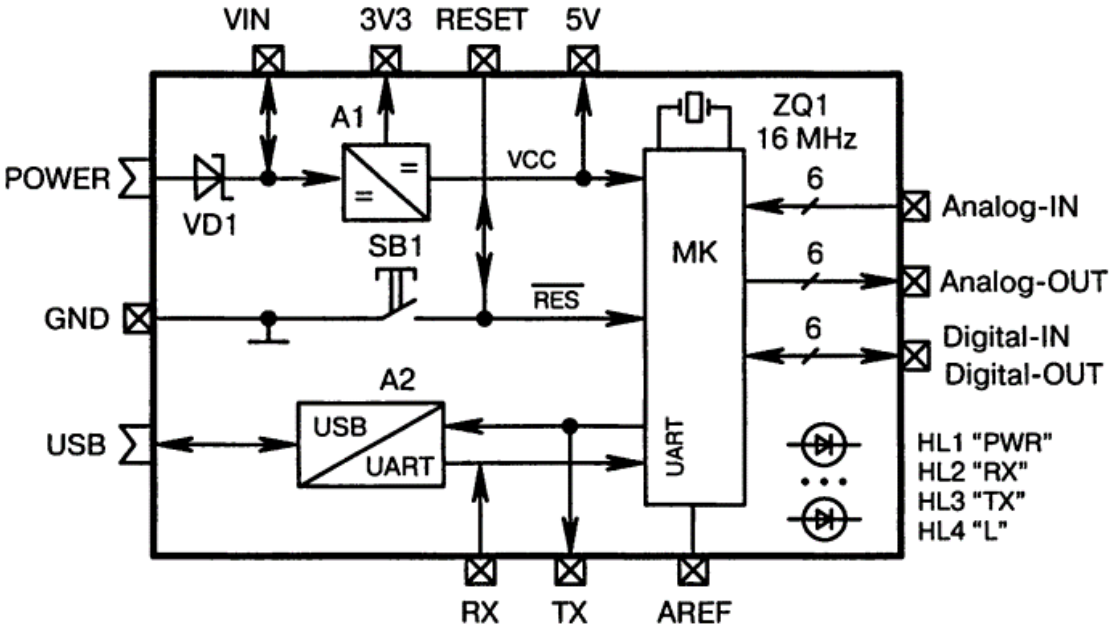


Рис.1.7. Структурна схема Arduino-UNO

Зв'язок з комп'ютером здійснюється через конвертор USB-UART. Живлення на нього та на Arduino 5В надходить від комп'ютера. Також передбачено зовнішнє живлення через роз'єм POWER від «мережевої вилки» з напругою 9... 12 В. Система живлення Arduino-UNO показана на рис.1.8.

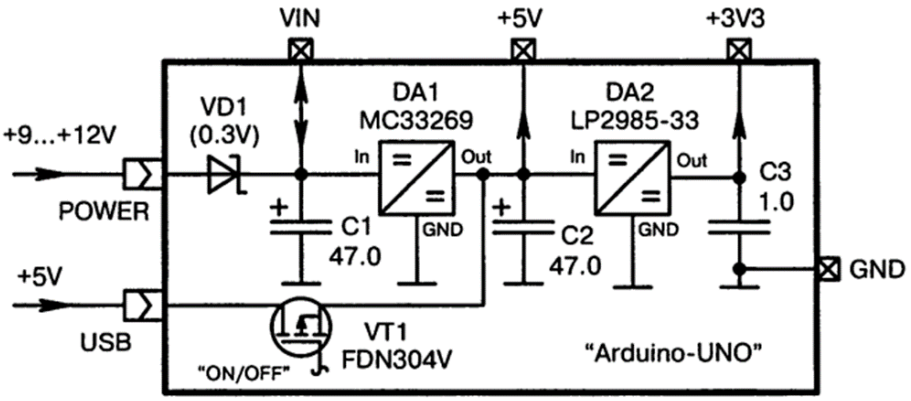


Рис.1.8. Схема організації живлення в Arduino-UNO

Вхідні і вихідні сигнали МК поділяються за функціональною ознакою на такі групи: цифрові входи (IN), цифрові виходи (OUT), аналогові входи (АЦП), аналогові виходи (ШІМ). Схемотехніка підключення зовнішніх вузлів по входу та виходу буде такою ж, як і для звичайних МК.

Кількість програмно доступних портів в Arduino не є константою. В залежності від налаштувань регістрів допускається різне поєднання цифрових та аналогових входів та виходів (рис.1.9). Лінії портів Arduino прийнято називати «пінами» (pin). В Інтернеті часто застосовуються терміни «порт», «лінія». Нумерація пінів наскрізна. Цифрові піни позначаються літерою «D», аналогові піни. літерою «A». Цифрові піни D0 та D1 відіграють особливу роль. Вони налаштовані на канал UART, який може в будь-який час необхідний для передачі даних в комп'ютер при налагодженні програми.

Розробники не встановлюють обмежень на тип МК, застосовуваний у Arduino, тому в його численних клонах використовують 8...32-бітні AVR, PIC-ARM-, Cortex-контролери. Кількість пінів (портів вводу/виводу) також може відрізнятися. Але для всіх моделей Arduino залишається незмінним принцип поділу портів на цифрові та аналогові, на входи та виходи.

Контакти «гребінки» 5V та 3V3 захищені від короткого замикання на загальний провід GND внутрішніми стабілізаторами DA1, DA2.

Плата Arduino NANO. це пристрій на основі мікроконтролера ATmega328 (рис.1.10). До його складу входить все необхідне для зручної роботи з мікроконтролером: 14 цифрових входів / виходів (з них 6 виводів

можуть використовуватися як ШІМ-виходи), 6 аналогових входів, кварцовий резонатор на 16 МГц, роз'єм USB, роз'єм живлення, роз'єм для внутрішньо схемного програмування (ICSP) та кнопка скидання.

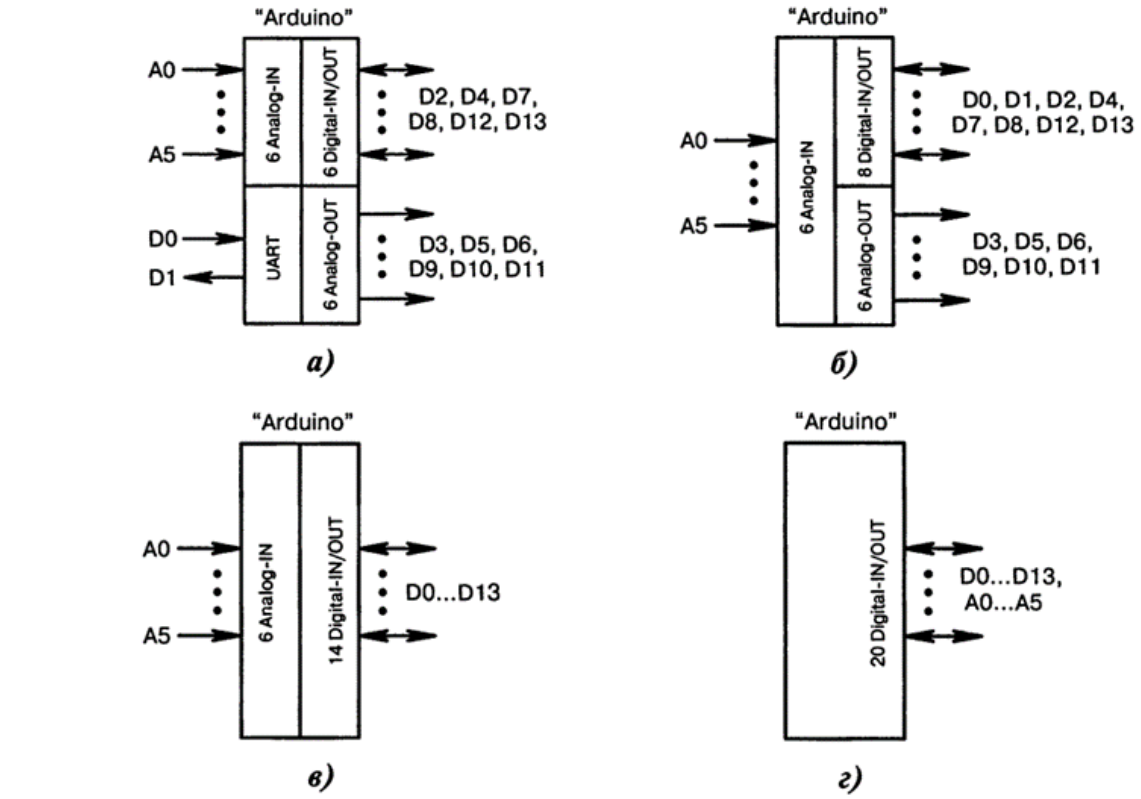


Рис.1.9. Конфігурація пінів Arduino: а) повна; б) без сигналів UART: в) без аналогових виходів; г) без аналогових входів

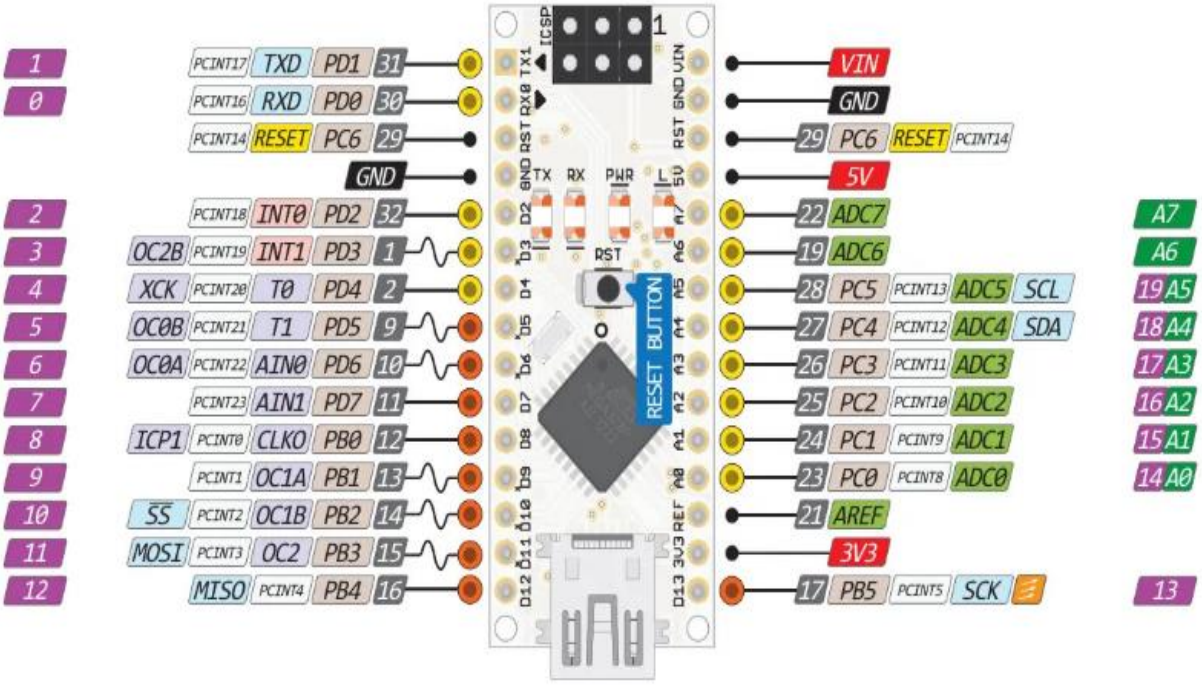


Рис.1.10. Призначення виводів плати Arduino NANO

Для початку роботи з платою необхідно подати живлення від AC / DC-адаптера, або підключити його до комп'ютера за допомогою USB-кабелю. Для роботи з платою Arduino Nano в операційній системі Windows необхідно встановити на комп'ютер інтегроване середовище розробки Arduino IDE (Integrated Development Environment).

Входи і виходи плати Arduino NANO:

- послідовний інтерфейс: виходи 0 (RX) і 1 (TX), що використовуються для отримання (RX) та передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними виводами мікросхеми ATmega8U2 на платі Arduino Nano, яка виконує роль перетворювача USB-UART;
- зовнішні переривання: виводи 2 і 3, які можуть служити джерелами переривань, що виникають при фронті, спаді або при низькому рівні сигналу на цих виводах.
- ШІМ-виводи 3, 5, 6, 9, 10 і 11. інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK);
- світлодіод є вбудованимим, приєднаний до виводу 13. Рівень HIGH включає світлодіод, LOW. виключає;
- 6 аналогових входів (A0 - A5), кожен з яких може представити аналогову напругу у вигляді 10-бітного числа. За замовчуванням, вимір напруги здійснюється у діапазоні від 0 до 5 В;
- Reset: формування низького рівня (LOW) на цьому виводу призведе до перезавантаження мікроконтролера. Зазвичай цей вивід служить для функціонування кнопки скидання на платах розширення;
- 3V3: напруга на даному виводі +3.3 В, що генерується вбудованим регулятором на платі. Максимальне споживання струму 50 мА. Від цього виводу можуть живитися деякі апаратні модулі;
- 5V: напруга на даному виводу +5 В, що генерується вбудованим регулятором на платі Arduino NANO;
- GND: земля, або загальний мінус;

- Vin: використовується для подачі живлення від зовнішнього джерела в відсутності живлення 5 В від роз'єму USB, наприклад, коли необхідно запустити плату Arduino окремо від комп'ютера;
- TWI / I2C: вивід A4 або SDA та вивід A5 або SCL;
- AREF: опорна напруга для аналогово-цифрового перетворювача.

Рівень напруги на виводах обмежений 5В, максимальний струм, який може віддавати або споживати один порт, становить 40 мА. Всі виводи пов'язані з внутрішніми резисторами номіналом 20-50 кОм.

1.3 Система команд і програмна модель AVR

Процес написання програм для МК AVR як і для будь-яких інших, складається з декількох етапів: підготовка вихідного тексту програми на якій-небудь мові програмування; компіляція програми; налагодження й тестування програми; остаточне програмування й підготовка до серійного виробництва.

Мікропрограма пристрою повинна бути написана на одній з мов програмування. На даний час для МК AVR існують декілька мов програмування, а також різних засобів підтримки розробки, що використовують одну мову, але різняться за функціональністю. На кожному з етапів необхідне застосування спеціальних програмних й апаратних засобів. Варто відзначити, що базовий набір програмного забезпечення (компілятор асемблера, ПЗ для програмування) поширюється фірмою Atmel безкоштовно. Однак за досить довгий період часу, що пройшов з моменту появи цих МК, з'явилася велика кількість програмного забезпечення сторонніх виробників [2].

Етапи розробки програмного забезпечення мікроконтролерів AVR наведені на рис.1.11.

Інтегроване середовище розробки Arduino IDE. це кросплатфомовий додаток на Java, що включає в себе редактор коду, компілятор та модуль передачі прошивки в плату.

Рис.1.11. Етапи розробки програмного забезпечення мікроконтролерів AVR

Щоб почати використовувати Arduino IDE, треба зайти на сайт https://www.arduino.cc, перейти на вкладку SOFTWARE > DOWNLOADS та завантажити середовище розробки Arduino (рис.1.12).

Рис.1.12. Завантаження середовища розробки

Далі необхідно інсталювати драйвер для плати Arduino, з якою треба працювати (це робить тільки з підключеною платою через USB):

1) Підключіть плату до комп'ютера і дочекайтеся, поки Windows не почне процес установки драйверів. Незважаючи на всі зусилля системи, через кілька секунд процес завершиться невдачею.
2) Зайдіть в Пуск, відкрийте Панель керування.
3) У Панелі керування перейдіть на сторінку Система і безпека. Далі клацніть по пункту Система і відкрийте Диспетчер пристроїв.
4) Знайдіть розділ Порти (COM & LPT). У ньому ви побачите відкритий порт під ім'ям «Arduino UNO (COMxx)».
5) Клацніть правою кнопкою по пункту «Arduino UNO (COMxx)» і виберіть «Оновити драйвер».
6) Далі, у вікні, виберіть пункт «Виконати пошук драйверів на цьому комп'ютері»
7) На завершення, виберіть файл драйвера під ім'ям «*arduino.inf*», розташований в папці «Drivers» в директорії завантаженого програмного забезпечення Arduino. Windows завершить інсталяцію драйвера.
8) Відкрити середовище розробки Arduino та запустити тестову програму File > Examples > 1.Basics > Blink.
9) Тепер в меню Tools>Board необхідно вибрати пункт меню, що відповідає вашій моделі Arduino.
10) У меню Tools>Serial Port виберіть послідовний порт, до якого підключений лабораторний макет. Як правило, це COM-порт з номером 3 (COM3) або вище (COM1 і COM2 зазвичай асоційовані з апаратними портами). Тепер можна працювати та завантажувати скетчі в Arduino.

Середовище розробки Arduino (рис.1.13) складається із вбудованого текстового редактора програмного коду, області повідомлень, вікна виведення тексту (консолі), панелі інструментів з кнопками часто

використовуваних команд і декількох меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини Arduino.

Програма, написана в середовищі Arduino, називається скетч. Скетч пишеться в текстовому редакторі, що має інструменти: вирізати / вставити, пошук / заміна тексту. Під час збереження та експорту проекту в області повідомлень з'являються пояснення, також можуть відображатися виникли помилки. Вікно виведення тексту (консоль) показує повідомлення Arduino, що включають повні звіти про помилки та іншу інформацію. Кнопки панелі інструментів дозволяють перевірити та записати програму, створити, відкрити та зберегти скетч, відкрити моніторинг послідовної шини.

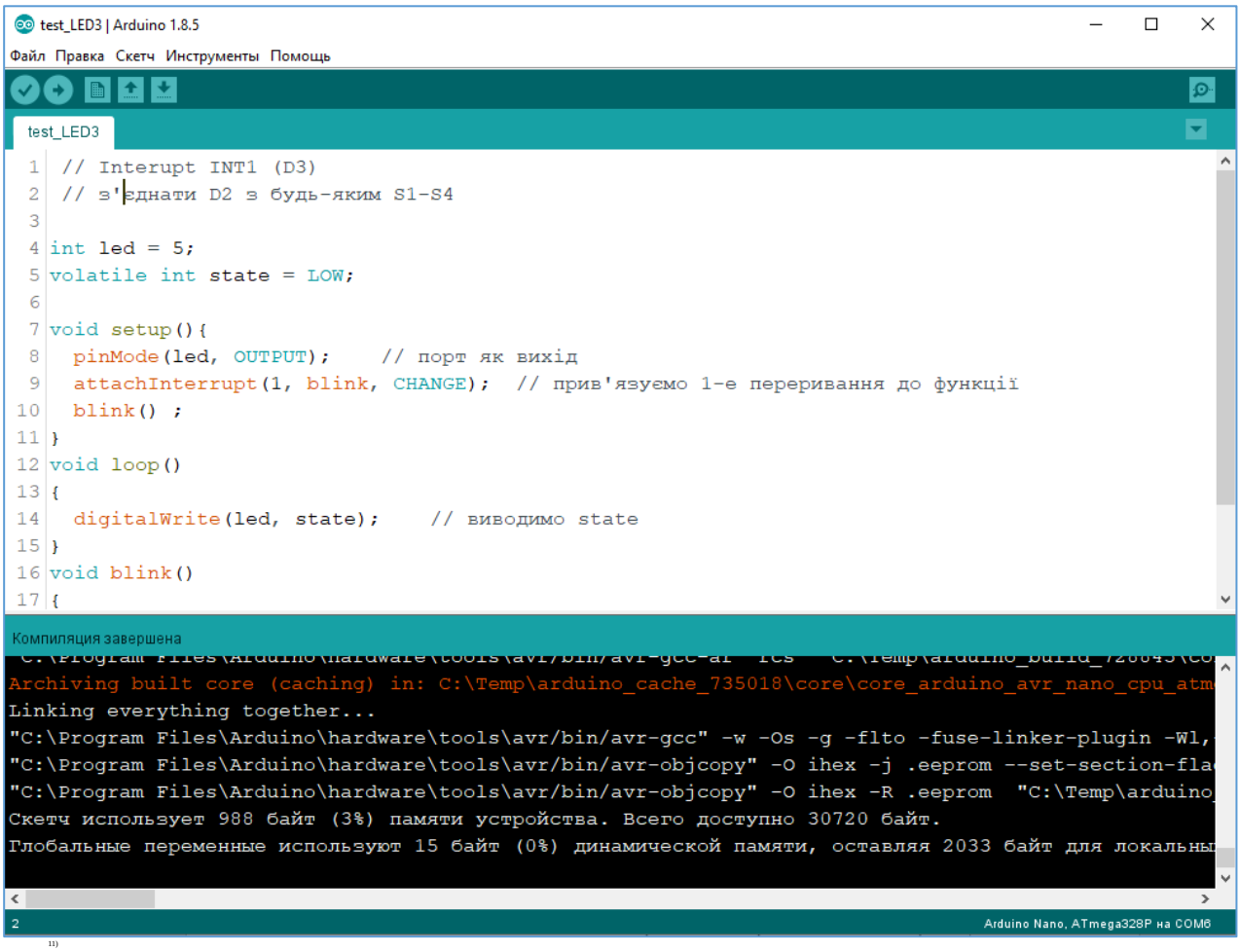


Рис.1.13. Загальний вигляд програми Arduino IDE

Блокнот (Sketchbook)

Середовищем Arduino використовується принцип блокнота: стандартне місце для зберігання програм (скетчів). Скетчі з блокнота

відкриваються через меню File → Sketchbook або кнопкою Open на панелі інструментів. При першому запуску програми Arduino автоматично створюється директорія для блокнота. Розташування блокнота змінюється через діалогове вікно Preferences.

Закладки, Файли і Компіляція

Дозволяють працювати з декількома файлами скетчів (кожен відкривається в окремій закладці). Файли коду можуть бути стандартними Arduino (без розширення), файлами C (розширення * .c.), файлами C ++ (*.cpp) або файлами заголовків (.h).

Завантаження скетчу в Arduino

Після вибору порту та платформи необхідно натиснути кнопку завантаження на панелі інструментів або вибрати пункт меню File → Upload to I/O Board. Сучасні платформи Arduino перезавантажуються автоматично перед завантаженням. На більшості плат під час процесу будуть мигати світлодіоди RX і TX. Середовище розробки Arduino виведе повідомлення про закінчення завантаження або про помилки.

При завантаженні скетчу використовується завантажувач (Bootloader) Arduino, невелика програма, що завантажується в мікроконтролер на платі. Вона дозволяє завантажувати програмний код без використання додаткових апаратних засобів. Завантажувач (Bootloader) активний протягом декількох секунд при перезавантаженні платформи і при завантаженні будь-якого з скетчів в мікроконтролер. Робота завантажувача (Bootloader) розпізнається по миготінню світлодіода (13 вивід) (наприклад, при перезавантаженні плати).

Експорт бінарного файлу

Для роботи з віртуальним стендом «Arduino Learner Kit» у середовищі Proteus 8 необхідно підключити файл типу .HEX (рис.1.14). Для його отримання потрібно вибрати пункт меню Sketch →Export Compiled Binary. HEX файл буде створений у директорії зі скетчим.

Бібліотеки

Бібліотеки додають додаткову функціональність скетчам, наприклад, при роботі з апаратною частиною або при обробці даних. Для використання бібліотеки необхідно вибрати меню Sketch → Include Library. Можна вибрати бібліотеки зі списку або завантажити через Library Manager (рис.1.14).

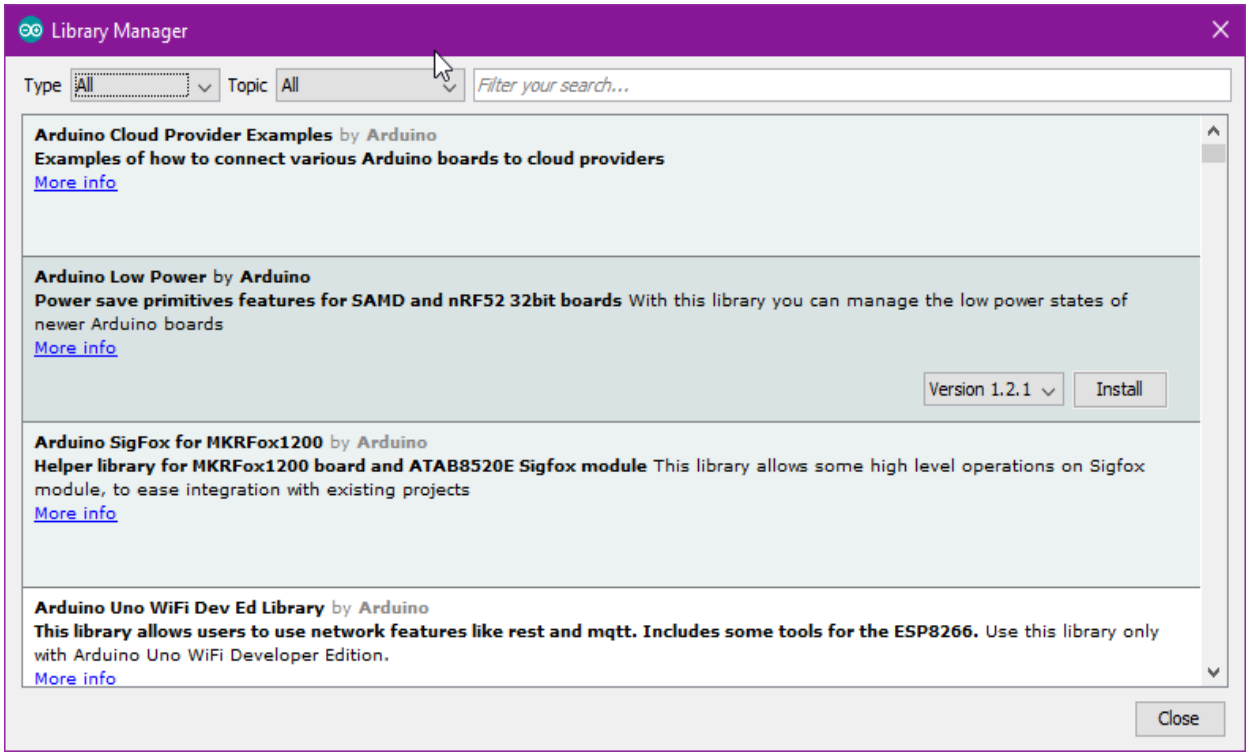


Рис.1.14. Менеджер завантаження бібліотек

Одна або кілька директив #include будуть розміщені на початку коду скетчу з подальшою компіляцією бібліотек разом зі скетчем. Завантаження бібліотек вимагає додаткового місця в пам'яті Arduino. Невикористані бібліотеки можна видалити з скетчу прибравши директиву #include.

Основні бібліотеки: EEPROM, SD, SPI, SoftwareSerial, Wire. Деякі бібліотеки включені в середовище розробки Arduino. Інші можуть бути завантажені з різних ресурсів. Для встановлення додаткових бібліотек необхідно створити директорію «libraries» у папці блокнота та потім розпакувати архів. Наприклад, для встановлення бібліотеки DateTime її файли повинні знаходиться в папці / libraries / DateTime папки блокнота.

1.4 Програмування в машинних кодах

Середовище розробки засновано на мові програмування Processing і спроектовано для програмування новачками, які не знайомі близько з розробкою програмного забезпечення. Строго кажучи, це C/C++, доповнений деякими бібліотеками. Програми обробляються за допомогою препроцесора, а потім компілюються за допомогою AVR-GCC.

Мова Arduino має чотири складових: оператори, дані, функції, бібліотеки.

Оператори:

- setup (),
- loop (),
- оператори мови C.

Дані: типи даних з мови C.

Функції:

- цифрове введення / виведення,
- аналогове введення / виведення,
- час,
- математичні обчислення,
- тригонометрія,
- випадкові числа,
- біти і байти,
- зовнішні переривання,
- переривання.

Бібліотеки:

- EEPROM,
- SD,
- SPI,
- SoftwareSerial,

– Wire,
– допоміжні класи,
– клас Serial,
– клас Stream.

Оголошення змінної відбувається так: спочатку вказується тип даних для змінної, а потім назва змінної. Оператор присвоювання (=). не є знаком рівності та не може використовуватися для порівняння значень. Оператор рівності записується так. ==. Присвоєння використовується для збереження певного значення в змінній. Наприклад, запис виду *a = 10* задає змінній *a* значення числа *10*.

Якщо знаємо точну кількість дій (ітерацій) циклу, то можемо використовувати цикл *for*. Синтаксис його виглядає так:

```
for (дія до початку циклу; умова продовження циклу; дія в кінці кожної ітерації циклу)
    {
        інструкція циклу 1;
        інструкція циклу 2;
        інструкція циклу N;
    }
```

Ітерацією циклу називається один прохід цього циклу.

Коли не відомо скільки ітерацій повинен зробити цикл, тоді використовується цикл *while* або *do ... while*. Синтаксис циклу *while* виглядає так:

```
while (Умова){
    Тіло циклу;}
```

Даний цикл буде виконуватися, поки умова, вказана в круглих дужках є істиною.

Оператор if служить для того, щоб виконати будь-яку операцію в тому випадку, коли умова є вірною. Умовна конструкція завжди записується в круглих дужках після оператора *if*. Усередині фігурних дужок вказується тіло умови. Якщо умова виконається, то почнеться виконання всіх

команд, які знаходяться між фігурними дужками.

Оператор else. Кожному оператору *if* відповідає тільки один оператор *else*. Сукупність цих операторів. *else if* означає, що якщо не виконалася попередня умова, то перевірити дану. Якщо жодна з умов не є вірною, то виконується тіло оператора *else*.

Оператори порівняння

== (дорівнює);

! = (не дорівнює);

< (менше ніж);

> (більше ніж);

<= (менше або дорівнює);

> = (більше або дорівнює).

Логічні оператори

&& (І);

|| (АБО);

! (НЕ);& (побітове І);

| (побітове АБО).

Бітові оператори

^ (побітове XOR або виключне АБО);

~ (побітове НЕ);

<< (побітовий зсув вліво);

>> (побітовий зсув вправо).

Складні оператори++ (інкремент)

–. (декремент);

+ = (складене додавання);

– = (складене віднімання);

* = (складене множення);

/ = (складене ділення);

& = (складене побітове І);

| = (складене побітове АБО).

Будь-яка функція має тип, як і будь-яка змінна. Функція може повертати значення, тип якого аналогічний типу самої функції. Якщо функція не повертає ніякого значення, то вона повинна мати тип *void* (такі функції іноді називають процедурами).

При оголошенні функції, після її типу має стояти ім'я функції і дві круглі дужки. відкриваюча і закриваюча, всередині яких можуть знаходитися один або кілька аргументів функції, яких також може не бути взагалі. Після списку аргументів функції ставиться відкриваюча фігурна дужка, після якої знаходиться саме тіло функції. В кінці тіла функції обов'язково ставиться фігурна дужка, що закриває його.

Під час виклику функції *setup(),* програма ініціалізується та встановлює початкові значення. Функція *setup()* викликається, коли стартує скетч. Використовується для ініціалізації змінних, визначення режимів роботи виводів, запуску використовуваних бібліотек. Функція *setup()* запускається тільки один раз, після кожної подачі живлення або скидання плати Arduino.

Функція *loop ()* забезпечує нескінченний робочий цикл програми. У циклі виконується опитування стану виводів, зміна їх стану, прийом-передача даних робота з АЦП та ін.

Приклад:

```
    int  buttonPin  =
3; void setup()
    {
    // put your setup code here, to run once:
    }
    void loop()
    {
    // ...
    }
```

Типи даних

void. ключове слово void використовується тільки при оголошенні функцій. Воно вказує на те, що оголошувана функція не повертає ніякого значення.

boolean. змінні типу boolean можуть приймати одне з двох значень: true або false. Кожна змінна типу boolean займає в пам'яті один байт.

char. тип даних, який займає в пам'яті 1 байт та зберігає символьне значення. Символи пишуться в одинарних лапках, наприклад: 'A' (сукупність символів (рядки) пишуться у подвійних лапках: "ABC").

int. цілочисельний тип даних. Це основний тип даних для зберігання чисел. В Arduino Uno змінні типу int зберігають 16-бітові (2-байтові) значення у діапазоні від.32768 до 32767.

unsigned int. беззнакові цілі містять двобайтові значення. Замість негативних чисел зберігаются лише позитивні значення в діапазоні від 0 до 65535.

long. змінні типу long маю розширений розмір для зберігання чисел та мають розмірність 32 біта (4 байта), що дозволяє їм зберігати числа в діапазоні від.2 147 483 648 до 2 147 483 647.

unsigned long. мають розмірність 32 біта (4 байта). Змінні типу *unsigned long*, на відміну від звичайного long, зберігають тільки позитивні числа в діапазоні від 0 до 4 294 967 295.

short. це 16-бітний тип даних.

float. тип даних для чисел з плаваючою точкою. Числа з плаваючою точкою часто використовуються для подання аналогових або безперервних величин, оскільки дають можливість окреслити їх більш точно, ніж цілі числа. Числа з плаваючою точкою мають 32 біта (4 байта) інформації та можуть досягати величезних значень від. 3.4028235E + 38 до 3.4028235E + 38.

Точність дрібних чисел типу *float* становить 6-7 десяткових знаків. Мається на увазі загальна кількість цифр, а не кількість знаків після коми.

double. займають 4 байта. Аналогічні змінним float.

Створення (оголошення) *масиву*. Масив . це область пам'яті, де можуть послідовно зберігатися кілька значень.

```
int myInts[6];
```

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

string. текстовий рядок. Може бути оголошений двома способами: можна використовувати тип даних *string* або оголосити рядок як *масив символів char* з нульовим символом в кінці.

```
char Str1 [15];
char Str2 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '\
0'};
char Str4 [] = "arduino";
char Str5 [8] = "arduino";
char Str6 [15] = "arduino".
```

Рядки завжди оголошуються в подвійних лапках ("Abc"), а символи завжди оголошуються в одинарних лапках ('A').

1.5 Порти введення/виведення AVR. Програмне введення/виведення інформації

За замовчуванням усі порти Arduino визначаються як входи, і немає потреби описувати це в коді. Порти зазвичай прописуються в функції ініціалізації змінних.

Ініціалізація порту вводу-виводу Arduino:

pinMode **(pin, mode).** Параметри:

– pin: номер виводу, режим роботи якого задається;

– mode: приймає значення: INPUT. вхід, у цьому режимі відбувається зчитування даних з датчиків, стану кнопок, аналогового та цифрового сигналу. Порт знаходиться в так званому високоімпедансному стані, тобто на вході високий опір. OUTPUT. вихід, залежно від команди прописаної в коді, порт приймає значення одиниці або нуля. Вихід стає свого роду керованим джерелом живлення та видає максимальний струм (20 мА та 40

мА в піковому значенні) у навантаження, що до нього підключене. INPUT_PULLUP. порт працює як вхід, але до нього підключається «PushUp» резистор з номіналом 20. 50 кОм;

– значення, що повертаються. немає.

digitalWrite **(pin, value).** Параметри:

– pin: номер виводу;

– value: значення HIGH або LOW;

– значення, що повертаються. немає.

digitalRead **(pin).** Параметри:

– pin: номер цифрового виводу, з якого необхідно зчитати значення (int);

– значення, що повертаються HIGH або LOW.

Регістри портів дозволяють низько рівневі 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.1.10). 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;
```

1.6 Таймери/лічильники. Модуль переривань

У ATMega328 передбачені три таймери/лічильники, на яких реалізовано функції часу, які використовують для формування та вимірювання часових інтервалів.

Основні функції часу:

delay (ms) Параметри:

- ms. кількість мілісекунд, на які необхідно призупинити програму;
- значення, що повертаються. немає;
- опис: припиняє виконання програми на вказаний проміжок часу (в мілісекундах).

delayMicroseconds (us). Параметри:

- us - кількість мікросекунд, на які необхідно призупинити програму;
- значення, що повертаються. немаєопис: припиняє виконання програми на вказаний проміжок часу (в мікросекундах). Найбільше число для

формування затримки. 16383.

millis ():

- параметри. немає;
- значення, що повертаються. кількість мілісекунд, що пройшли з моменту старту програми;
- опис: повертає кількість мілісекунд, що пройшли з моменту старту програми Arduino. Число, що повертається, скинеться в 0) через приблизно 50 днів.

micros():

- параметри. не має;
- значення, що повертаються. кількість мікросекунд, що минули з моменту старту програми;
- опис: повертає кількість мікросекунд, що минули з моменту початку виконання програми Arduino. Число, що повертається, скинеться в 0 через приблизно 70 хвилин. Роздільна здатність цієї функції становить чотири мікросекунди.

pulseIn (pin, value) / pulseIn (pin, value, timeout). Параметри:

- pin : номер виводу, на якому буде очікуватися сигнал;
- value: тип імпульсу (HIGH або LOW);
- timeout (опціонально): час очікування імпульсу в мікросекундах (значення за замовчуванням - одна секунда);
- значення, що повертаються. тривалість імпульсу (в мікросекундах) або 0 в разі відсутності імпульсу протягом таймаута;
- опис: зчитує тривалість імпульсу (будь-якого. HIGH або LOW) на виведення. Наприклад, якщо задане значення value = HIGH, то функція *pulseIn ()* очікує появи на виведення сигналу HIGH, потім вимірює час та чекає перемикання в стан LOW, після чого зупиняє відлік часу. Функція повертає тривалість імпульсу в мікросекундах, або 0 в разі відсутності імпульсу протягом певного часу очікування. Функція працює з імпульсами тривалістю від 10 мікросекунд до 3 хвилин.

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.

Як входи для прийому запитів від зовнішніх джерел, найчастіше використовуються виводи паралельних портів вводу/виводу, для яких ця функція є альтернативною. Джерелами запитів зовнішніх переривань також можуть бути будь-які зміни зовнішніх сигналів на деяких спеціально виділених лініях портів вводу/виводу.

Arduino надає свої функції для роботи з зовнішніми перериваннями. Ці функції оголошені у файлі:\ *Hardware \ cores \ arduino \ wiring.h* та реалізовані в файлі: \ *Hardware \ cores \ arduino \ WInterrupts.c*

Функції переривання Arduino:

attachInterrupt (interrupt, function, mode). Параметри:

- interrupt: номер переривання (0 - pin D2, 1- pin D3);
- function: функція, яку необхідно викликати при виникненні переривання; ця функція повинна бути без параметрів і не повертати ніяких значень (таку функцію іноді називають оброблювачем переривання);
- mode: визначає умову, за якої має спрацьовувати переривання. Може приймати одне з чотирьох визначених значень: LOW. переривання буде спрацьовувати щоразу, коли на виводі присутній низький рівень сигналу, CHANGE. переривання буде спрацьовувати щоразу, коли змінюється стан виводу, RISING. переривання спрацює, коли стан виводу зміниться з низького рівня на високий, FALLING. переривання спрацює, коли стан виводу зміниться з високого рівня на низький;

– значення, що повертаються. немає.

detachInterrupt (pin):

– параметри. немає;

– значення, що повертаються. немає;

– опис: забороняє задане переривання. Забороняє переривання, для того, щоб відключити доступ до даних в процесі виконання переривання.

interrupts ():

– параметри. немає;

– значення, що повертаються. немає;

– опис: повторно дозволяє переривання.

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

Для обробки переривань у мові програмування Arduino використовується функція ISR ().

1.7 Мікроконтролери Arduino та ESP8266

Будь-який розробник мікропроцесорних систем управління використовує мікроконтролери Arduino (UNO, Micro, Nano, Mini, Mega), ESP8266 (WeMos D1, WeMos D1 mini NodeMCU), AirBoard, ChipKIT (UNO32, DP32, uC32, Max32) [9, 10]. Для зв'язку між контролером та мобільним пристроєм можна скористатися Bluetooth HC-05, HC-06, WiFi ESP8266, Ethernet Shield W5100. Для створення пристрою з віддаленим керуванням пропонуються кілька

безкоштовних систем розробки та використання мобільних графічних інтерфейсів для управління контролерами зі смартфона або планшета.

Основним елементом керування є модуль з мікроконтролером, який здійснює приймання та обробку команд по каналу Wi-Fi і керує виконавчим пристроєм.

Блок керування можна реалізувати такими способами. Перший спосіб зробити зв'язку на Arduino Uno та Wi-Fi шилд на базі HDG04 [9, 10]. Апаратне зв'язування даних модулів відбувається накладанням Wi-Fi шилда на Arduino Uno, у вигляді «бутерброда».

Wi-Fi шилд побудований на базі модуля HDG104, представляє собою систему на кристалі, яка забезпечує підключення Arduino до мережі Інтернет по безпровідному інтерфейсу LAN 802.11b/g (Wi-Fi). Мікроконтролер ATmega 32UC3 підтримує стек мережевих протоколів (IP) та дозволяє працювати як з TCP, так і з UDP-протоколами. Дана збірка має сильні недоліки, а саме: велика ціна збірки Arduino Uno + Wi-Fi шилд.

Другий спосіб зробити зв'язку з Arduino Pro Mini nf ESP-01(ESP-12E). В даній зв'язці в ролі мікроконтролера виступає Arduino Pro Mini, а в ролі пристрою передачі через інтернет виступає ESP-01(ESP8266). Оскільки на виході мікроконтролера рівень виходів дорівнює 5В, в пристрої перетворення рівнів немає необхідності.

Третій спосіб використати модуль Node MCU V3, який виступає у ролі пристрою передачі даних через Інтернет і мікроконтролером керування. Технічні характеристики модуля Node MCU V3 наведені в [10].

Node MCU V3 [10]. це модуль, на якому розташований чіп ESP 8266, який містить Wi-Fi мікросхему і мікроконтролер. Завдяки цьому можна створювати мікропроцесорні системи інтернету речей, що з'єднуються між собою через Wi-Fi з'єднання. Характерними особливостями цього модуля є частота мікроконтролера, яка складає 160МГц, велика кількість портів введення/виведення мале енергоспоживання і компактні розміри. Під даний

модуль можна писати програму на мові програмування Processing та JavaScript.

Wi-Fi модуль розроблений компанією Ai-thinker і побудований на базі процесора з ядром ESP8266, відмінною рисою якого є наявність радіоінтерфейсу Wi-Fi. Ядро ESP8266 інтегровано в Tensilica L106 - 32-бітний мікроконтролер з ультранизьким енергоспоживанням. Підтримка тактових частот 80 і 160 МГц, підтримка RTOS, вбудовані Wi-Fi MAC / BB / RF / PA / LNA. Флеш-пам'ять, інтегрована в модуль - це SPI флеш-пам'ять, ємність якої становить 4 Мбайта, в корпусі SOP-210mil. Антена, що застосовується в модулі, мікросмугова антена на платі з коефіцієнтом посилення 3 дБ. Однокристальна Wi-Fi система ESP8266EX вбудовується разом з контролером пам'яті, включаючи SRAM і ROM. MCU може звертатися до пам'яті через інтерфейси iBus, dBus і AHB. Розмір RAM <36 Кбайт, тобто, коли ESP8266EX працює в режимі клієнтської станції і підключений до роутера, програмований простір, доступний користувачеві разом з секцією Data, становить близько 36 Кбайт. В однокристальній системі немає програмованої пам'яті ROM; призначена для користувача програма повинна зберігатися у зовнішньому SPI флеш-пам'яті.

Модуль містить 11 портів загального призначення. Деякі з портів мають додаткові функції: D9, D10 - UART, D1, D2 - I²C/TWI, D5..D8 - SPI, D1..D10 - виходи з ШІМ (PWM), A0 - аналоговий вихід з АЦП.

1.8 Аналого-цифрові перетворювачі. Цифро-аналогові перетворювачі

Для передавання аналогового сигналу до МПС використовують аналогово-цифровий перетворювач (АЦП). АЦП сприймає аналоговий сигнал, напругу або струм і перетворює його в цифрове слово, зрозуміле МП. На рис.1.15 наведена структурна схема модуля (АЦП), що застосовується в МК AVR.

Перетворення «аналог-цифра» здійснюється в 10-бітовому АЦП послідовного наближення. Для його нормальної роботи потрібно три сигнали: вхідний V_{IN}, тактовий $F_{АЦП}$, опорний V_{REF}.

Сигнал V_{IN} поступає від мультиплексора, що комутує вісім аналогових каналів з ліній PA0...PA7 та дві тестових напруги 0 і +1,22В. Вибір джерела сигналу здійснюється програмним способом через регістр ADMUX. Сигнал $F_{АЦП}$ виходить з тактового сигналу F_{CLK} шляхом ділення на коефіцієнт 2...128, що програмно задається регістром ADCSRA. Усередині блоку АЦП частота $F_{АЦП}$ ділиться ще раз на 13 або 14 залежно від одноразового або безперервного режиму вимірів. Це і буде істинним значенням частоти дискретизація сигналу Fд.

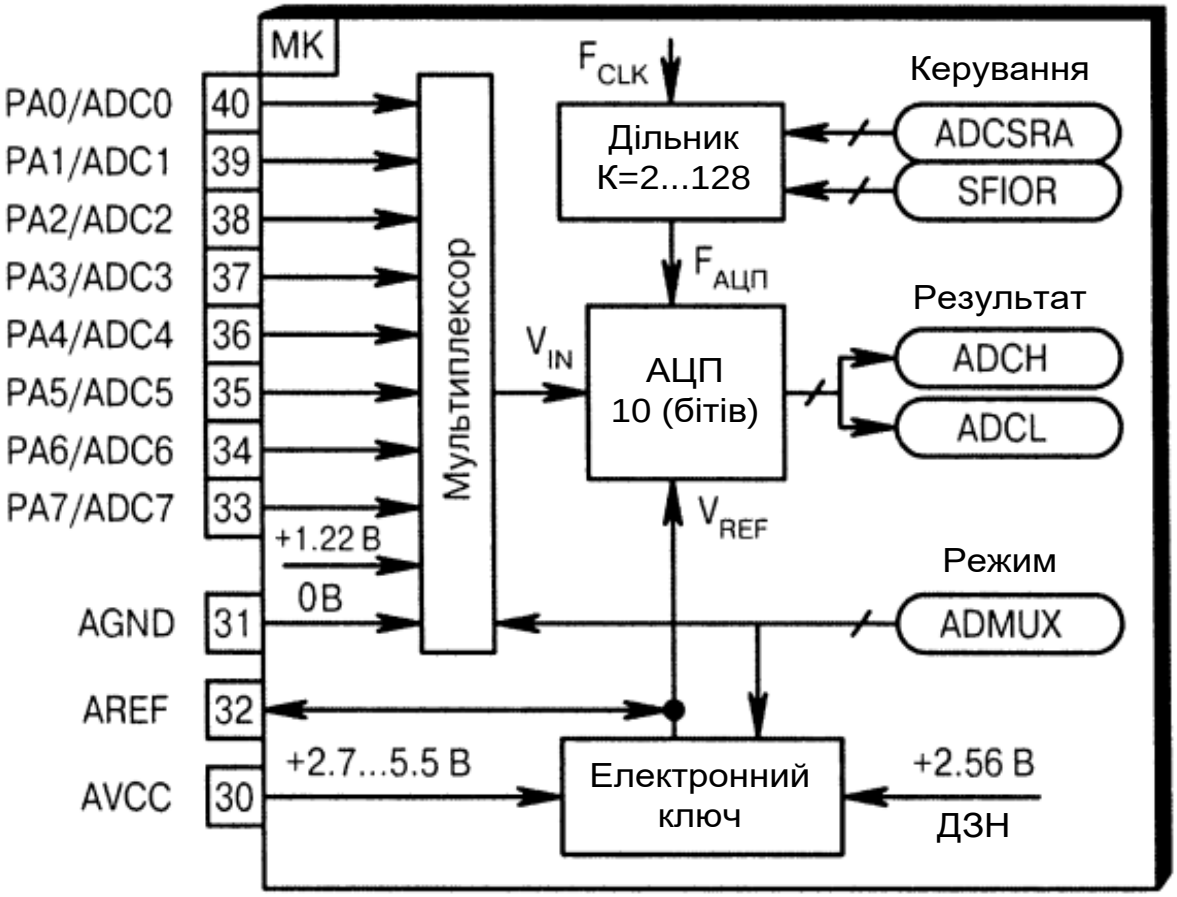


Рис.1.15. Структурна схема модуля АЦП в мікроконтролерах AVR

Припустимо, що F_{CLK} = 8 МГц, коефіцієнт ділення встановлений 16, відповідно, $F_{АЦП}$ = 500 кГц, Fд = 38,5 кГц (13тактів) або Fд= 35,7 кГц (14 тактів).

Сигнал V_{REF} може поступати з трьох напрямів: від вхідної лінії AREF, від внутрішнього джерела зразкової напруги +2,56 В, від джерела живлення AVCC. Перемикання здійснюється електронним ключем, який керується регістром ADMUX. Вивід AREF має безпосередній електричний зв'язок з модулем АЦП, тому, для зменшення наведень, його шунтують керамічним конденсатором ємністю 0,1 мкФ.

Результат кожного вимірювання поміщається в регістри ADCH (старші 2 біта) і ADCL (молодші 8 бітів). Разом в двох регістрах утворюється число в діапазоні 0... 1023. Ціна одного ділення. V_{REF}/1024. Це справедливо для «чистого» режиму 10 біт. У ATmega328 є ще один режим, що умовно називається 8/10 біт. У ньому вимірювання проводиться з точністю 10 бітів, але відображаються усього лише 8 старших бітів, тобто діапазон складає 0...255, ціна одного ділення V_{REF}/256.

При роботі АЦП потрібно на початку програми відключати вхідні «pull-up» резистори, оскільки лінії ADC0...ADC7 за сумісництвом ще є цифровими входами PA0...PA7.

Аналого-цифровий перетворювач дозволяє зчитати величину напруги на аналогових входах. Це дає можливість зчитувати дані з датчика освітленості, виміряти напругу живлення і т.д. Основні команди для роботи з АЦП в Arduino:

***analogReference* (type).** Параметри:

- type: тип джерела опорної напруги (DEFAULT, INTERNAL, EXTERNAL);
- значення, що повертаються. немає;
- опис: встановлює джерело опорної напруги, що використовується при зчитуванні аналогового сигналу (задає максимальне значення вхідного діапазону). DEFAULT: опорна напруга за замовчуванням, рівна 5В.

INTERNAL: внутрішня опорна напруга рівна 1,1В. EXTERNAL: як опорна напруга буде використовуватися напруга, прикладена до виводу AREF (від 0 до 5В).

analogRead (pin). Параметри:

- pin: номер виводу, з якого буде зчитуватися напруга (A0 - A5 для більшості плат, A0 - A7 для Mini та Nano, A0 - A15 для Mega);
- значення, що повертаються: ціле число int (від 0 до 1023);
- опис: зчитує величину напруги з зазначеного аналогового виводу. АЦП перетворювач перетворює вхідну напругу з діапазону 0 - 5В в цілочисельні значення в межах від 0 до 1023 відповідно. Роздільна здатність АЦП становить: 5 В / 1024 значення або 0,0049 В (4.9 мВ) на одне значення. Вхідний діапазон та роздільна здатність можуть змінюватися за допомогою функції *analogReference ()*. Для зчитування значення з аналогового входу потрібно близько 100 мікросекунд, тому максимальна частота опитування виводу приблизно дорівнює 10 000 разів в секунду. Якщо аналоговий вхід ні до чого не підключений, значення, що повертається функцією *analogRead (),* буде випадковим (змінюється під впливом декількох факторів: величина напруги на інших аналогових входах, наведення від руки поблизу плати).

Цифроаналоговий перетворювач (ЦАП) призначений для перетворення числа, визначеного, як правило, у вигляді двійкових кодів, у напругу або струм пропорційно значенню цифрового коду.

Дуже часто ЦАП входить до складу МПС. У цьому випадку, якщо не потрібна висока швидкодія, цифроаналогове перетворення може бути дуже просто здійснено за допомогою широтно-імпульсної модуляції (ШІМ). Схема ЦАП з ШІМ наведена на рис.1.16.

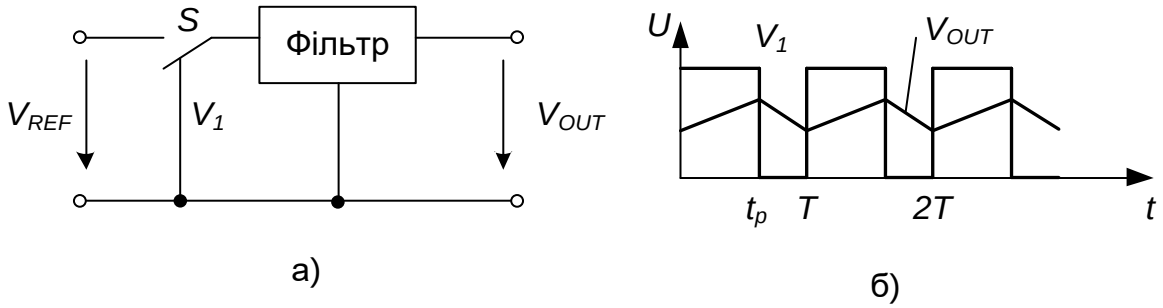


Рис.1.16. ЦАП з широтно-імпульсною модуляцією:

а) структурна схема; б) часова діаграма

Найпростіше організовується цифроаналогове перетворення в тому випадку, якщо МК має вбудовану функцію широко-імпульсного перетворення. Вихід ШІМ керує ключем *S*. У залежності від заданої розрядності перетворення контролер за допомогою власного таймера/лічильника формує послідовність імпульсів, відносна тривалість яких $\gamma = t_p / T$ визначається співвідношенням:

$$\gamma = \frac{D}{2^N},$$

де *N*. розрядність перетворення, а *D*. код перетворення.

Фільтр нижніх частот згладжує імпульси, виділяє середнє значення напруги. У результаті вихідна напруга перетворювача:

$$V_{OUT} = \gamma V_{REF} = \frac{DV_{REF}}{2^N}$$

Розглянута схема забезпечує ідеальну лінійність перетворення, не містить прецизійних елементів (за винятком джерела опорної напруги). Основний її недолік. низька швидкодія.

Для формування сигналів ШІМ використовується команда ***analogWrite (pin, value)*** з параметрами:

- pin: вивід, на якому буде формуватися напруга широтно-імпульсної модуляції (ШІМ або PWM);
- value: коефіцієнт заповнення. лежить в межах від 0 (завжди вимкнений)

до 255 (завжди включений);

- значення, що повертаються: немає;
- опис: формує задану аналогову напругу на виводі у вигляді ШІМ-сигналу. Може використовуватися для зміни яскравості світіння світлодіода або управління швидкістю обертання двигуна. Після виклику *analogWrite(),* на виводі буде безперервно генеруватися ШІМ-сигнал із заданим коефіцієнтом заповнення до наступного виклику функції *analogWrite()* (або до моменту виклику *digitalRead(),* або *digitalWrite(),* взаємодіючих з цим же виводом). Частота ШІМ становить приблизно 490 Гц. На більшості плат Arduino функція *analogWrite ()* працює з виводами 3, 5, 6, 9, 10 і 11. Функція *analogWrite ()* не має нічого спільного з аналоговими виводами і функцією *analogRead ().*

РОЗДІЛ 2. ФУНКЦІОНАЛЬНІ ВУЗЛИ МІКРОПРОЦЕСОРНИХ СИСТЕМ

2.1 Особливості живлення та формування тактової частоти

Для живлення будь-якого МК потрібно, як мінімум, два виводи: позитивний («плюс», «Power supply») і негативний («мінус», «Ground reference»). Позначають їх в Data-sheet і на схемах: Vcc (Voltage Collector - to - Collector) або V_{DD} (Voltage Drain - to - Drain); GND (Ground) або V_{SS} (Voltage Source - to - Source).

Двопровідне живлення застосовується в малогабаритних МК з числом виводів 6... 18, наприклад, в Atmel ATtiny, Microchip PIC10/12. З розвитком технології до складу МК стали вводити аналогові вузли АЦП/ЦАП, які дуже чутливі до завад. Додавання ланок AVCC (Analog VCC) і AGND (Analog GND) дозволяє розв'язати між собою аналогові і цифрові частини мікросхеми, зменшити імпульсні завади, підвищити інструментальну точність каналів АЦП і ЦАП.

Якщо подивитися на осцилограму струму споживання МК, то в ній можна помітити низькочастотну (НЧ) і високочастотну (ВЧ) складові. Як наслідок, коливання струму призводять до появи НЧ- і ВЧ - завад на клемах живлення. Для їх послаблення використовують стандартні рішення у вигляді зв'язки конденсаторів (рис.2.1), LC - і RC - фільтрів (рис.2.2).

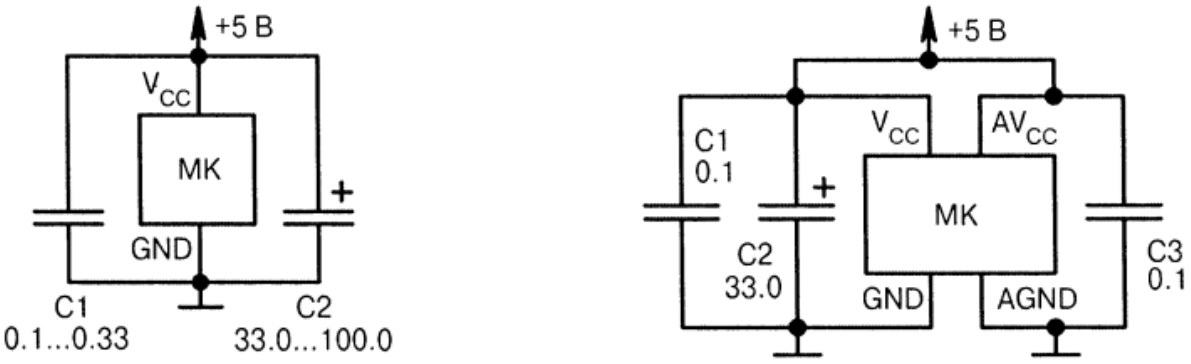


Рис.2.1. Фільтрація завад в схемі живлення

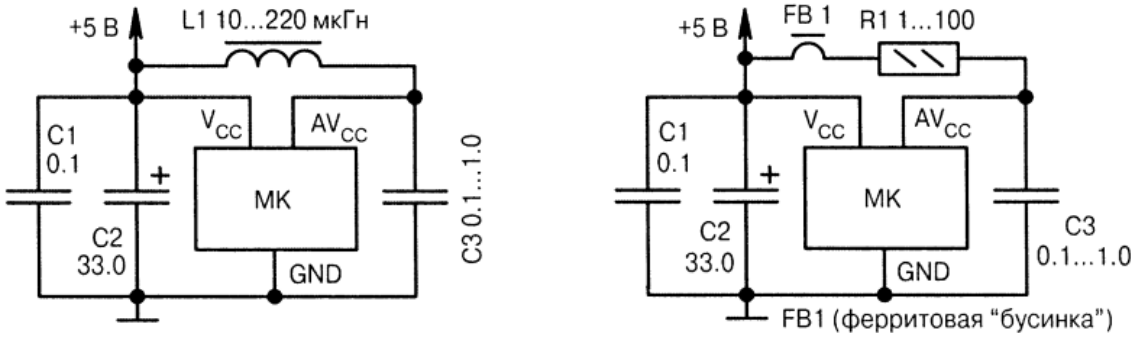


Рис.2.2. Фільтрація завад LC-фільтром та RC-фільтром

Неполярні конденсатори С1, С3 (рис.2.1) послабляють ВЧ-завади. Їх наявність обов'язкова біля будь-якого МК, причому максимально близько від виводів живлення (не більше 50 мм). Конденсатори мають бути керамічні, наприклад, К10-17 або SMD типу 0603... 1206. Базовий номінал ємності 0,1 мкФ вибраний умовно, тому що його легко запам'ятати. Полярний конденсатор С2 бажано використати танталовий, оскільки він краще подавлює імпульсні завади. При виборі ємності можна керуватися таким правилом. 1000 мкФ на кожен ампер струму навантаження. Наприклад, якщо цифрова частина МК споживає струм 10...30 мА, то досить поставити конденсатор С2 ємністю 10...30 мкФ з робочою напругою не менше 6,3В. Рекомендують вибирати конденсатори з напругою 10... 16 В, оскільки підвищується надійність в експлуатації і, головне, знижується внутрішній імпеданс, що дозволяє краще фільтрувати завади.

Конденсатор С2 обов'язковий при живленні від акумулятора як накопичувач енергії, а також при значних коливаннях і скачках напруги. У деяких випадках його функцію виконує конденсатор фільтру мережевого випрямляча або стабілізатора напруги.

Котушка індуктивності L1 (рис.2.2) розв'язує цифрову і аналогову частини по високій частоті. Якщо її не ставити, то може зменшитись точність виміру АЦП і стабільність порогу спрацьовування аналогового компаратора. Значну частину завад по живленню створюють внутрішні цифрові вузли МК, тому LC - і RC - фільтри захищають контролер від самого себе. Номінал

індуктивності L1 не особливо критичний і може змінюватися в широких межах.

Феритова «бусинка» FBI (Ferrite Bead) є провідник, пропущений через феритове кільце або циліндр. Цей елемент сприяє зниженню високочастотного випромінювання.

МК складається із статичних тригерів, регістрів і лічильників. Після подання живлення їх потрібно примусово встановити в певний логічний стан, інакше із-за загального хаосу виконання програми стане непередбачуваним. Імпульс початкового скидання подається на виводи RST (ReSeT) або RES (інверсний RESet). Відрізняються вони між собою, відповідно, позитивною і негативною формою сигналу.

Початковий скид у сучасних МК проводиться в таких випадках:

- Power-On. внутрішне автоматичне скидання, яке активізується відразу після подачі живлення;
- Brown-Out. скидання від внутрішнього детектора «просідань» напруги живлення;
- External Reset. зовнішній скид низьким рівнем на виводі RES;
- Watch-Dog. скидання від внутрішнього «сторожового» таймера при випадковій зупинці роботи центрального процесора або зависанні програми;
- JTAG - програмний скид через налагоджувальний інтерфейс JTAG.

Усі джерела скидання рівноцінні. Установка режимів скидання виконується бітами конфігурації, а також програмно-доступними регістрами з області SFR. Настроюватися можуть: поріг спрацьовування детектора напруги, що «просіла», тривалість часу затримки таймера очікування Watch - Dog.

Вузол апаратного скидання Power - On є присутнім в усіх без виключення МК. Якщо напруга живлення стабільна в часі і подається різким стрибком, то зовнішні елементи для скидання теоретично взагалі не потрібні. Скидання виконується автоматично вузлом Power - On при досягненню певного порогу.

При високій швидкості наростання живлення (орієнтовно за час не більше 1...5 мс) вхід скидання RES підключають до кола VCC трьома способами: безпосередньо, через зовнішній резистор опором 1-10 кОм або залишають вільним, покладаючись на внутрішній резистор МК.

Перший варіант повністю усуває шлях перешкодам, але виключає скидання кнопкою і можливість повторного програмування. Другий варіант дозволяє підключити кнопку скидання, що зручно при лабораторному макетуванні в домашніх умовах. Третій варіант дозволяється за відсутності перешкод і наявності усередині МК резистора підтяжки R_{RES} опором до 100 кОм.

Якщо напруга живлення наростає більше тривалий час (для різних сімейств МК по-різному), то рекомендується установка зовнішніх RC -ланок (рис.2.3, а, б) із стандартними значеннями:

- R1 = 10 кОм, C1 = 0,1 мкФ при часі наростання 5...20 мс. Наприклад, подача живлення Vcc перемикачем, який розташовується між інтегральним стабілізатором напруги +5 В і МК;
- R1 = 10 кОм, C1 = 10мкФ при часі наростання 20...100 мс. Це актуально, наприклад, при включенні пристрою в мережу 220 В загальним тумблером. Якщо ємність конденсатора C1 більше 1 мкФ, то для прискорення його розряду ставлять діод VD1 типу 1N4148 (КД522Б), а для захисту входу скидання від перенапруги ще і резистор R2. Ці перестраховки подовжують МК життя.

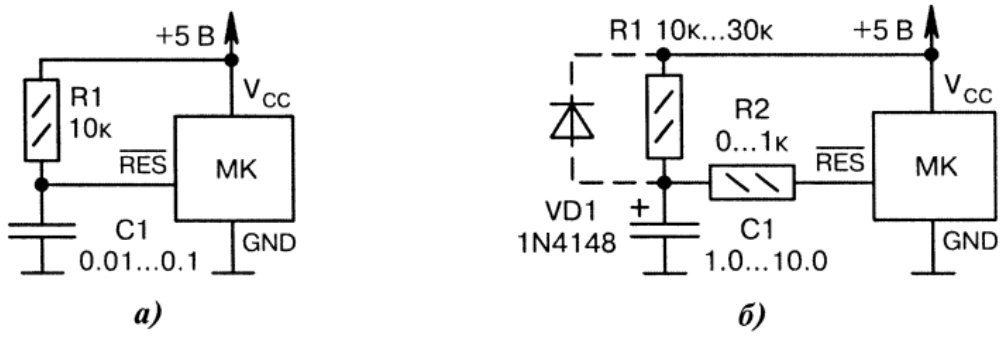


Рис.2.3. Скидання зовнішньою RC ланкою: а) при середньому часу наростання живлення; б) при великому часі наростання живлення

Вибір конкретної схеми скидання залежить від умов експлуатації. Наприклад, якщо сумарна ємність конденсаторів фільтру між VCC і GND складає більше 1000 мкФ, то, швидше за все, знадобиться зовнішня RC. ланка (рис.2.3, а, б). Якщо поряд з виводом RES на друкованій платі проходить силове комутаційне коло, то для з'ясування причин збоїв корисно тимчасово з’єднати лінію скидання МК з живленням. Якщо прилад розташовується поблизу від джерела потужних індустріальних завад, то на вході скидання рекомендується поставити додаткову мікросхему супервізора живлення, яка продублює вузол Brown - Out.

Для того, щоб МК запрацював, необхідно подати на центральний процесор тактові імпульси. Чим вище їх частота, тим швидше виконуються операції, а чим нижче їх частота, тим менше споживання струму. Формуванням тактових частот займається підсистема синхронізації. На її структурній схемі (рис.2.4) є декілька вбудованих генераторних вузлів (on - chip oscillator): HF(High Frequency) - високочастотний, LF (Low Frequency) - низькочастотний, CLK (CLocK) - тактування.

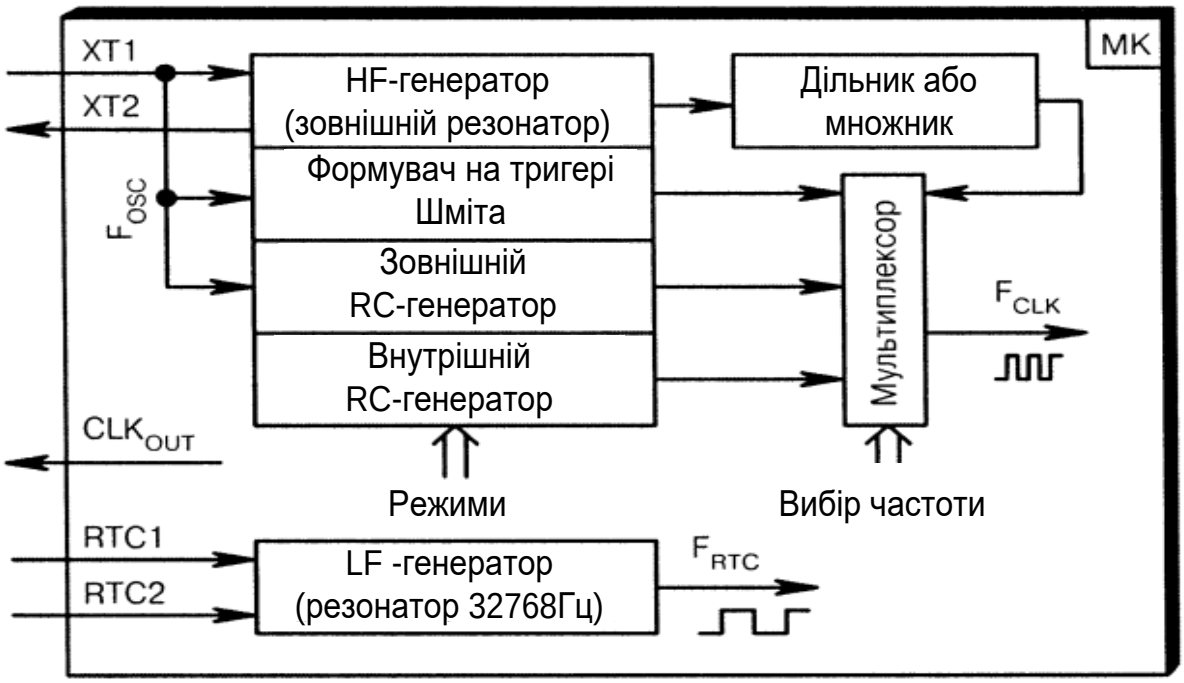


Рис.2.4. Підсистема синхронізації МК

З чотирьох верхніх блоків, тільки HF-генератор використовує два виводи підключення XT1, XT2. Тим самим підкреслюється, що він розрахований на схему із зворотним зв'язком.

Виводи XT1, XT2 основного генератора в різних МК можуть позначатися по-різному: XTAL1, XTAL2, XI, X2, XIN, XOUT, OSC1, OSC2.

Управління режимами підсистеми синхронізації здійснюється через біти конфігурації. Вони перемикають канали мультиплексора, настроюють частоту внутрішнього RC-генератора і так далі. Вони ж можуть дозволити/заборонити видачу сигналу CLKOUT (рис.2.4) з окремої лінії порту, з частотою, у декілька разів менше тактовою. Ця функція є присутньою не в усіх МК.

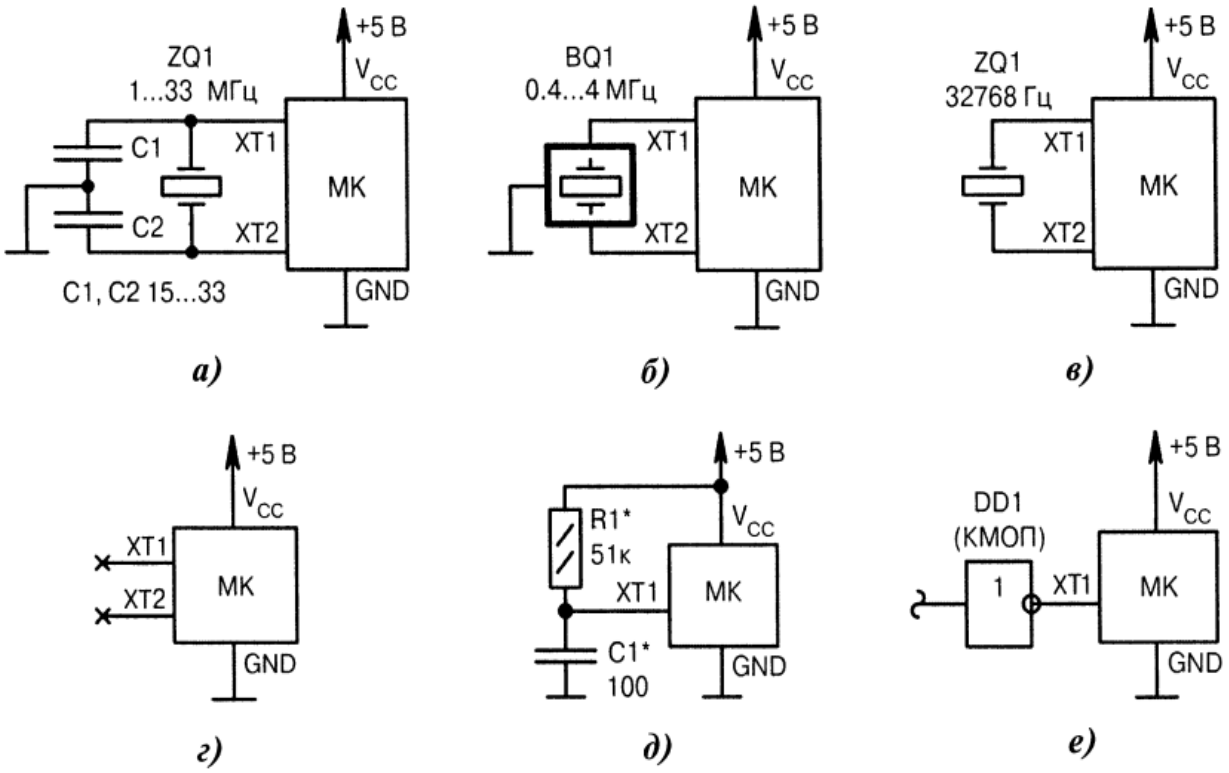


Рис.2.5. Схеми формування тактової частоти : а) від ВЧ кварцового резонатора; б) від керамічного резонатора; в) від НЧ кварцового резонатора; г) від внутрішнього генератора; д) від RC-ланки; е) від зовнішніх імпульсів.

МК працює в таких режимах:

- від високочастотного кварцового резонатора 1...33 МГц (рис.2.5, а);
- від середньо частотного керамічного резонатора 0,4...4 МГц (рис.2.5, б);

– від низькочастотного кварцового резонатора 10... 100 КГц (рис.2.5, в);
– від внутрішнього RC - генератора 1; 2; 4; 8 МГц (рис.2.5, г);
– від зовнішньої RC - ланки 0,4...12 МГц (рис.2.5, д);
– від зовнішніх синхроімпульсів 0...40 МГц (рис.2.5, е).

Усі перераховані режими роботи задаються при програмуванні конфігураційних бітів

2.2 Виконавчі пристрої мікропроцесорних систем управління

Практично жодна мікропроцесорна система управління не може обійтися без таких елементів, як виконавчі пристрої. Головне призначення будь-якої системи. це управління яким-небудь зовнішнім механізмом. Це можуть бути електродвигуни, нагрівачі, електромагнітні клапани. Тому, окрім датчиків, кнопок управління і елементів індикації до мікроконтролера обов'язково доведеться підключати і виконавчі пристрої. Для управління зовнішніми пристроями використовуються ті ж самі порти введення/виведення МК, які працюють на виведення. Сигнали з будь-якою з ліній будь-якого порту легко можуть бути використані для включення і виключення зовнішнього пристрою. Необхідно лише підсилити керуючий сигнал за потужністю до необхідного рівня. Для цього застосовуються різні схеми узгодження. Вибір схеми залежить від типу виконавчого пристрою.

У найпростішому випадку можна застосувати транзисторний ключ (рис.2.6). При використанні транзистора КТ315Г можна керувати зовнішніми колами із струмом споживання до 100 мА і напругою $U_Ж$ до 15 В. Транзистор допускає також високу напругу, проте підвищення напруги можливе при зменшенні струму.

Для керування ланками з великим струмом потрібно застосувати потужніший транзистор або цілу транзисторну збірку. При виборі транзистора потрібно враховувати, що максимально допустимий струм навантаження для будь-якого з виходів МК не повинен перевищувати величини 20 мА. При

складанні програми потрібно не забувати, що будь-який транзисторний ключ інвертує сигнал. Якщо на виході P1.0 (рис.2.6) встановити одиничний рівень, ключ відкривається і навантаження підключається до джерела живлення. При нульовому рівні на тому ж виході ключ закривається і навантаження відключається.

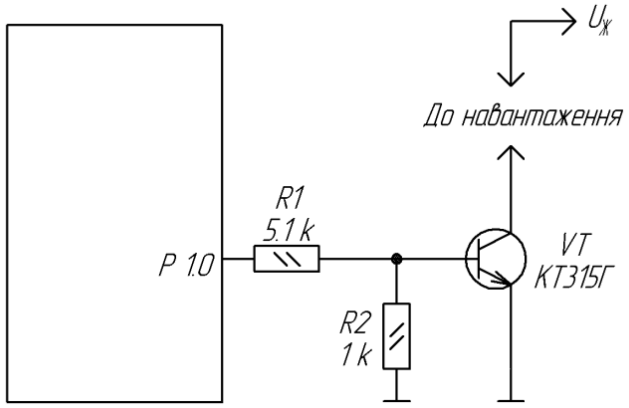


Рис.2.6. Найпростіший транзисторний ключ

Якщо виконавчий механізм, яким повинна керувати МПС, живиться від мережі змінного струму 220 В, потрібно застосовувати схему управління з гальванічною розв’язкою. Один з можливих варіантів. релейна схема управління.

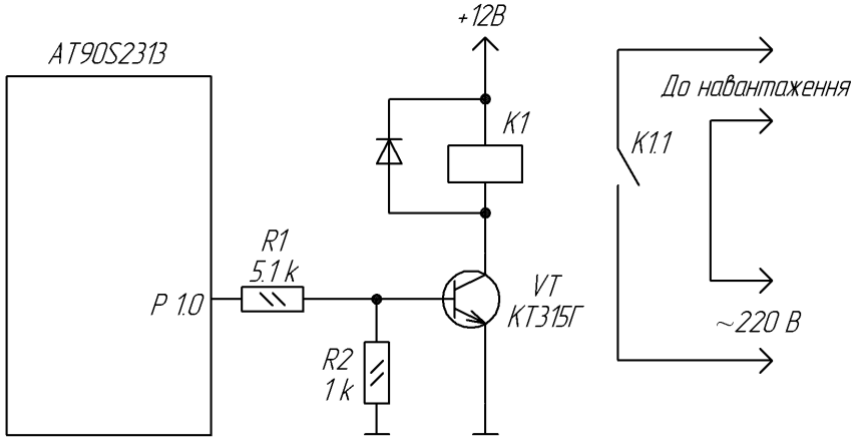


Рис.2.7. Виконавчий пристрій з використанням реле

Типовий варіант схеми управління з використанням реле наведений на рис.2.7. На схемі представлений електронний ключ, в навантаження якого включено електромагнітне реле К1. МК за допомогою ключа може вмикати і

вимикати електромагнітне реле. Контакти реле, в свою чергу, керують навантаженням. Така схема забезпечує комутацію достатньо великої напруги і струму.

Гальванічна розв'язка між всіма колами МПС і силовою мережею 220 В забезпечує безпеку роботи з цією схемою. Діод VD1 призначений для захисту елементів схеми від напруги ЕРС самоіндукції, що виникає в котушці К1 у момент закривання ключа VT1. При виборі електромагнітного реле необхідно звертати увагу на такі параметри. По-перше, напруга спрацьовування реле. Для прикладу на Рис.2.7 вона має бути рівна 12 В. По-друге, максимально допустимий струм комутації і максимально допустима напруга для виконавчих контактів реле. Вони повинні відповідати реальним значенням струму і напруги в колі навантаження.

Досить часто реле замінюють оптоелектронними комутаційними вузлами, які мають малі струми і напруги керування, беззвучні і довговічні у роботі, можливість робити в середовищах постійного і змінного струму, комутації напруги (деяких приладів) до 400...600 В і струмів до 0,5 А. На рис.2.8 представлена одна з таких схем.

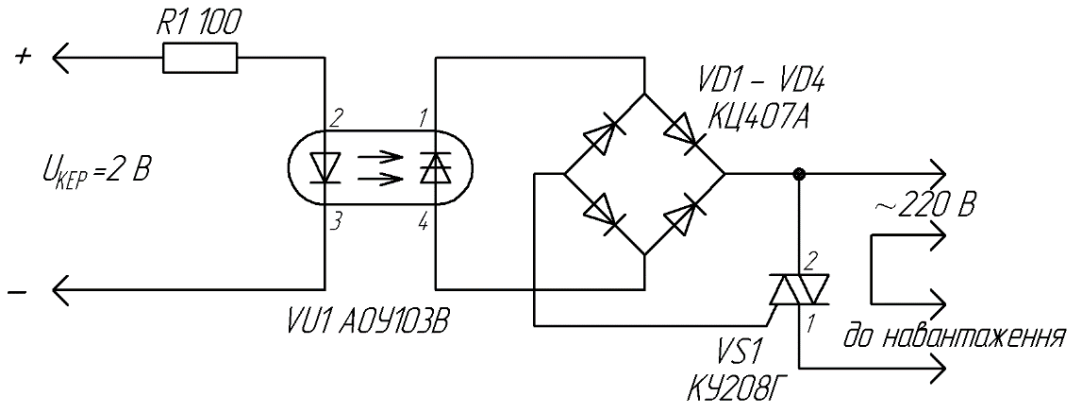


Рис.2.8. Електрична схема із застосуванням оптрона

У цій схемі управління навантаженням (потужність якого може досягати 600 Вт) здійснюється симістором КУ208. Завдяки розв'язці по живленню - застосуванню оптоелектронного приладу АОУ103В, кола управління навантаженням в мережі 220 В і керуючі схеми повністю розв'язані. Керуюча

постійна напруга (або імпульси) амплітудою 1,5...2 В потрапляє від схеми управління через обмежувальний резистор R1 на вхід оптопари VU1. Керуючий струм не перевищує 5 мА.

За наявності керуючого сигналу, тиристор усередині оптопари відкривається (його опір в прямому напрямку зменшується до декількох десятків Ом), і він шунтує діагональ випрямляючого моста VD1. Від випрямляючого моста напруга проходить на електрод керуючого симістора VST, завдяки чому він відкривається у відповідні напівперіоди напруги і в навантаженні тече струм. Використання оптопар АОУ103 залежить від напруги в електричному колі. Так, для даної схеми та інших з напругою більше 200 В підходить лише оптопара АОУ103В.

При необхідності управління більш потужним навантаженням, наприклад до 1000 Вт, симістор, як основний пристрій в даній схемі, що комутує навантаження, слід встановити на охолоджуючий радіатор.

Схожа за принципом роботи схема представлена на рис.2.9. Тут діагональ випрямного моста замикає оптосимістор ТО132-40 (або аналогічний ТО125-12,5, ТО106-10 та інші). Їх основна відмінність один від одного полягає у різних струмах і потужності комутації.

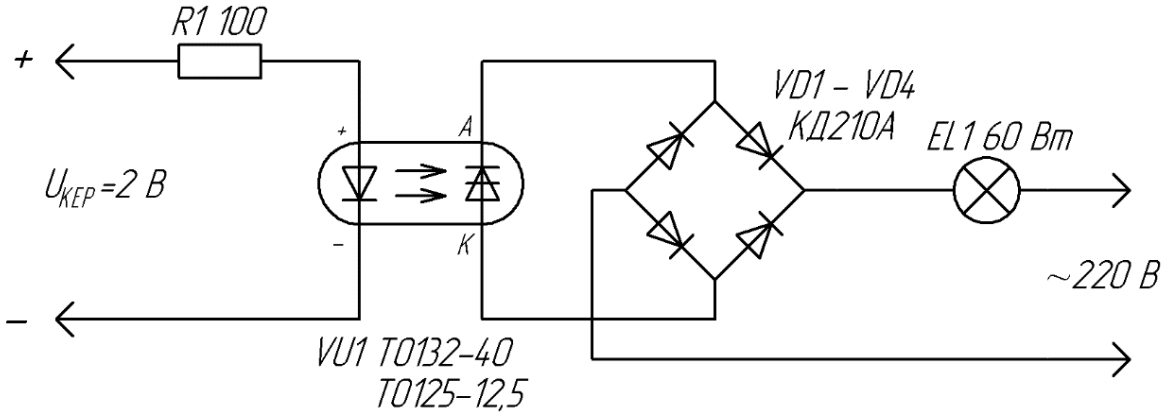


Рис.2.9. Схема вузла управління навантаженням із застосуванням оптрона

На рис.2.10 показаний ще один варіант включення - поєднання оптоелектронної розв'язки із застосуванням оптопари АОУ103В і симістора

КУ208Г.

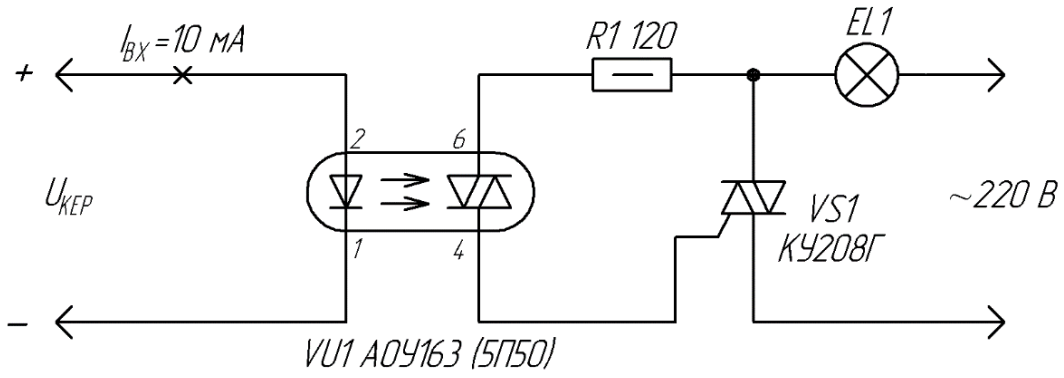


Рис.2.10. Електрична схема оптоелектронної розв'язки

Управління пристроями навантаження ефективно здійснюється, якщо їх потужність не перевищує 600 Вт. Оптопара АОУ103В дозволяє самостійно комутувати високовольтне навантаження (з напругою до 350 В), проте струм комутації не повинен перевищувати 100 мА. Тому для управління потужним навантаженням в схему введений симістор КУ208Г.

2.3 Елементи індикації

Практично кожна мікропроцесорна система управління містить елементи індикації. Як індикатори в даний час найчастіше застосовуються світлодіоди. На ринку є величезний вибір світлодіодів найрізноманітніших видів і розмірів.

У мікропроцесорній системі управління LED індикатори можуть служити для відображення різних режимів роботи: попередження про критичні ситуації, відображення ходу прийому сигналів керування тощо. Підключити одиночний світлодіодний індикатор до МК дуже просто. На рис.2.11 наведена схема підключення світлодіода безпосередньо до виводу порту МК.

Усі вихідні каскади МК мають достатню навантажувальну здатність для того, щоб витримати підключення одного світлодіодного індикатора зі споживаним струмом у робочому режимі не більше 20 мА.

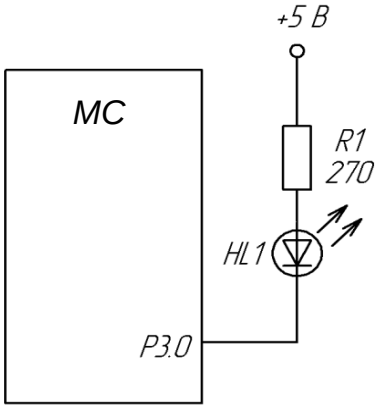


Рис.2.11. Схема підключення світлодіодного індикатора

Для керування двома світлодіодами одним виходом у МК передбачено активні вихідні каскади, і для перемикання режиму роботи (введення або виведення) слугує спеціальний регістр. Таким чином, сигнал кожного виходу будь-якого порту може мати 3 значення. "0", "1" і високоімпедансний ("Z") стан. Це дозволяє керувати двома світлодіодами за допомогою одного виводу (рис.2.12).

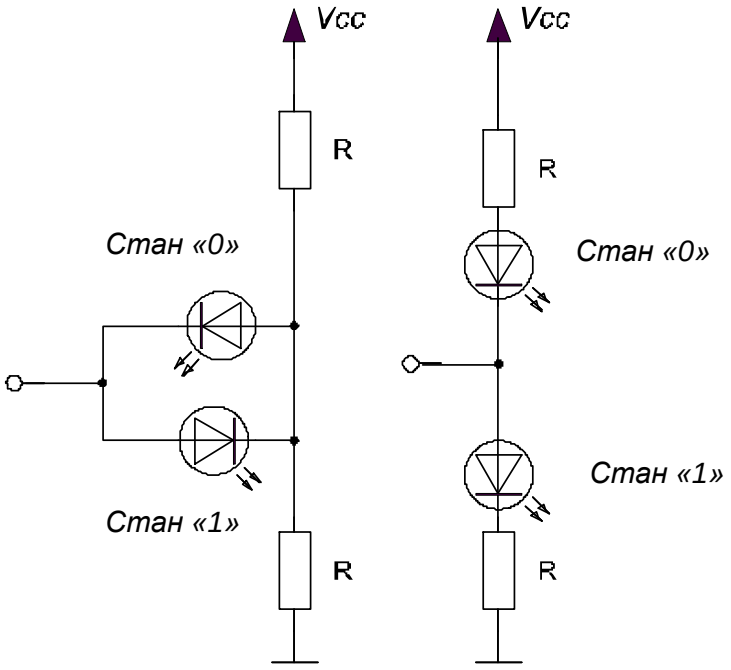


Рис.2.12. Керування двома світлодіодами одним виходом МК

При роботі порту в режимі виходу, залежно від стану "0" або "1" горить

відповідно верхній або нижній світлодіод. При перемиканні в Z-стан, і при відповідному виборі резисторів струм через світлодіоди дуже малий і їх світіння майже непомітно.

Дуже часто МК використовується не тільки для керування роботою, але й для того, щоб повідомити що-небудь користувачеві і/або отримати від нього які-небудь вказівки про подальшу роботу. Наприклад, електронний годинник, крім відліку часу, повинен його ще відображати, а також дозволяти змінювати покази (встановлювати точний час). Якщо вся «інформація» зводиться до мигання парою світлодіодів, яких-небудь спеціальних зусиль з відображення інформації з боку розробника не вимагається, але якщо таких світлодіодів виявляється два-три десятки, тут вже потрібне застосування додаткових засобів (як апаратних, так і програмних). Як правило, в цьому випадку відображення інформації виконують у режимі динамічний індикації. це найбільш економний за кількістю використовуваних ліній спосіб.

Для організації динамічної індикації застосовується матриця, що складається з ліній рядків і ліній стовпців (рис.2.13). На перетині стовпця і рядка матриці розташований індикаторний елемент. світлодіод. Для того, щоб запалити той або інший елемент, необхідно подати на матрицю не один, як у звичайних індикаторах, а два сигнали: логічна 1 на відповідному рядку і логічний 0 на відповідному стовпці матриці. Через односторонню провідність світлодіода кожна комбінація сигналів на входах рядків і стовпців однозначно включає рівно один індикаторний елемент.

Головна перевага динамічної індикації. невелике число ліній, що керують: для матриці світлодіодів розміром N×N елементів потрібно всього 2N сигналів, що керують. За таку економію, втім, доводиться платити. справа в тому, що при почерговому виведенні інформації на кожен світлодіод матриці його яскравість світіння буде в N^2 разів нижча, ніж при безпосередньому виведенні інформації на один світлодіод, що "окремо стоїть". Тому в пристроях, що використовують динамічну індикацію, виведення інформації здійснюється не на кожен світлодіод окремо, а на один рядок або на один

стовпець повністю. у цьому випадку яскравість світіння світлодіодів падає тільки в N разів.

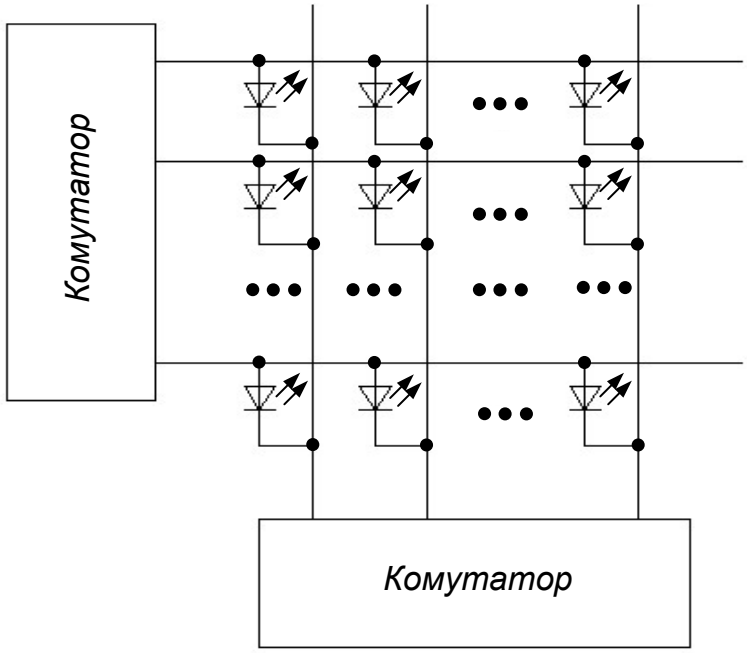


Рис.2.13. Організація динамічної індикації

Розглянемо декілька варіантів реалізації динамічної індикації із застосуванням МК [1, 2, 9, 10]. Як індикаторний елемент передбачається застосування семисегментних індикаторів, але кожен такий індикатор з легкістю можна замінити і групою світлодіодів.

Схема реалізації динамічної індикація без додаткових елементів наведена на рис.2.14.

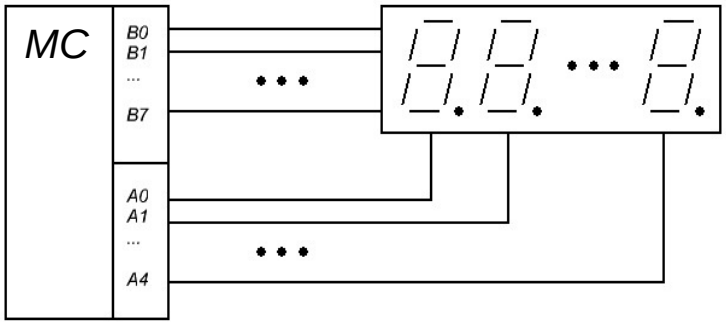


Рис.2.14. Схема реалізації динамічної індикація без додаткових елементів

До порту В МК підключені катоди всіх світлодіодів матриці, а до порту А. аноди кожного з індикаторів, що створюють матрицю. На лініях порту А організовується одиниця, що «біжить». На лінії порту В при кожному положенні одиниці, що біжить, виводиться семисегментний код того символу, який повинен горіти в даному знакомісці. Для індикаторів із загальним катодом замість одиниці, що біжить, використовується нуль, що біжить. Перевага такого способу індикації. у відсутності яких-небудь додаткових компонентів (окрім самих світлодіодних індикаторів), головний недолік. значна перевитрата ліній портів. Таке рішення для МК може забезпечити роботу не більше 5 семисегментних індикаторів одночасно. При використанні інших МК з великою кількістю ніжок вказана проблема знімається.

Схема реалізації динамічної індикації з одним додатковим елементом наведена на рис.2.15. У цьому варіанті одиниця, що біжить, реалізується за допомогою регістра зсуву. Порт В у цій схемі також використовується в режимі часового мультиплексування, як для видачі символу, так і для занесення чергового біта до регістра зсуву. Схема також вимагає всього одну додаткову лінію (крім порту В).

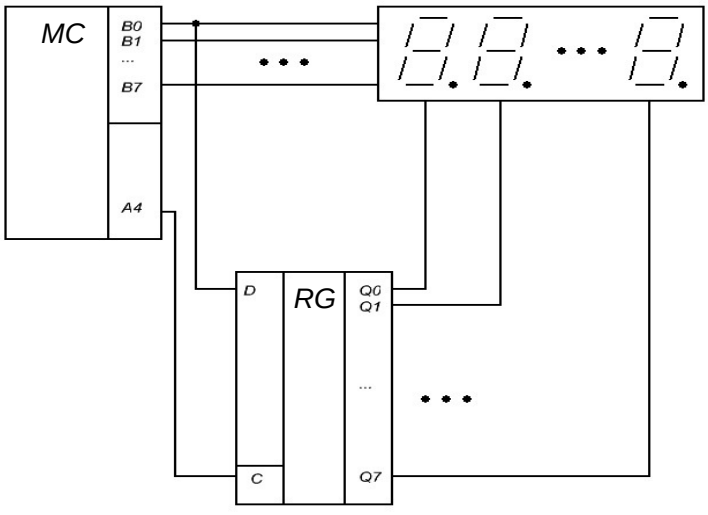


Рис.2.15. Схема реалізації динамічної індикації з регістром зсуву

Ще один варіант реалізації схеми з одним додатковим елементом наведений на рис.2.16. Така схема придатна тільки для індикаторів із

загальним катодом, оскільки для організації нуля, що біжить, в ній використовується дешифратор. Схема вимагає три додаткові лінії (крім порту B), проте у багатьох випадках вона може виявитися корисною.

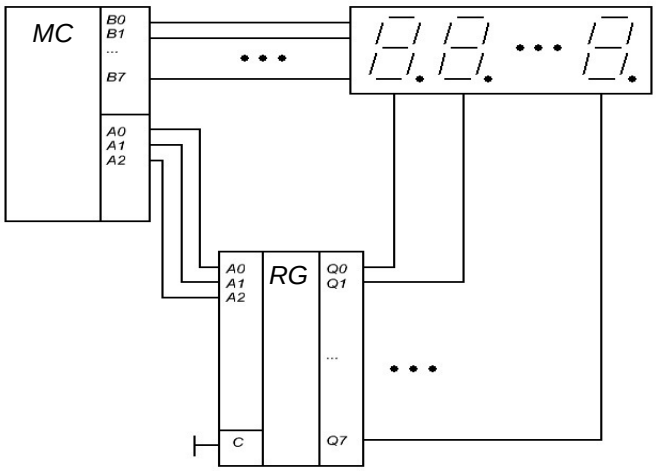


Рис.2.16. Схема реалізації динамічної індикації з дешифратором

2.4 Кнопки та датчики. Оптичні датчики

За допомогою кнопок і простих датчиків в мікропроцесорну систему управління надходить різна інформація, яка використовується для зміни алгоритму роботи програми. Схема підключення контактного датчика до МК наведена на рис.2.17. У наведеному прикладі датчик підключений до лінії PD0 порту D МК. Через цей вхід МК проводить зчитування стану датчика. Датчик можна підключити і до будь-якої іншої лінії будь-якого з портів МК.

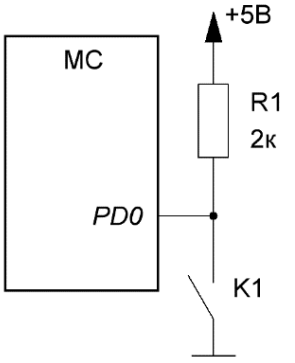


Рис.2.17 . Підключення контактного датчика до МК

У початковому стані контакти датчика розімкнені. На вхід МК через резистор R1 прикладається напруга від джерела живлення + 5 В. МК сприймає цю напругу як сигнал логічної одиниці. При спрацьовуванні датчика контакти замикаються і з’єднують вивід МК із загальним дротом. Тепер мікросхема сприймає вхідний рівень сигналу як логічний нуль. Резистор R1 при цьому служить струмообмежувальним елементом, запобігаючи короткому замиканню між шиною живлення і загальним дротом. Деякі МК мають свої внутрішні резистори навантаження, які можуть замінити зовнішній резистор. Схема підключення декількох датчиків або кнопок до МК зображена на рис.2.18.

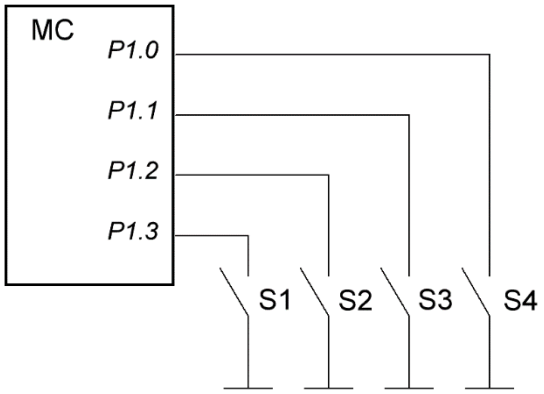


Рис.2.18. Підключення кнопок або простих датчиків до МК

У схемі, що зображена рис.2.19, при натисканні однієї з клавіш змінюється постійна напруга на відповідному вході процесора, яка розпізнається процесором і дешифрується в певну команду. Ця напруга максимальна (приблизно 5 В), коли кнопки не натиснуті, і мінімальна (0 В) при натиснутій клавіші S1 [1, 2, 10].

Існує два види клавіатур, що підключаються до МК: зі скануванням і з кодуванням.

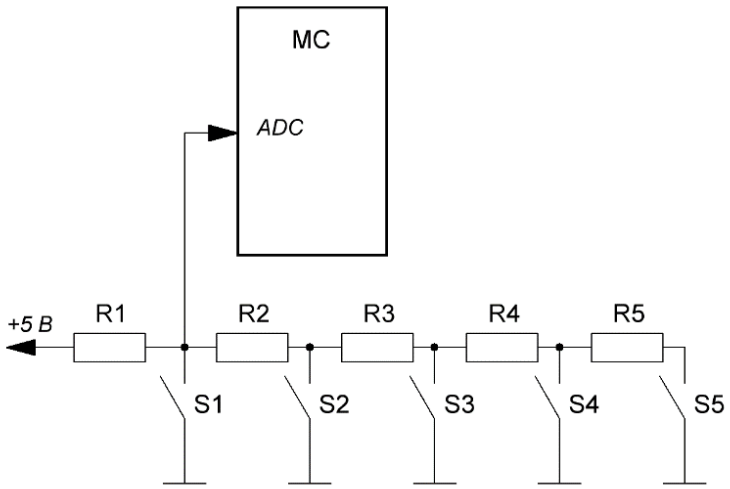


Рис.2.19. Підключення кнопок зміною напруги на аналоговому вході МК

Блок-схема 12-клавішної клавіатури зі скануванням показана на рис.2.20.

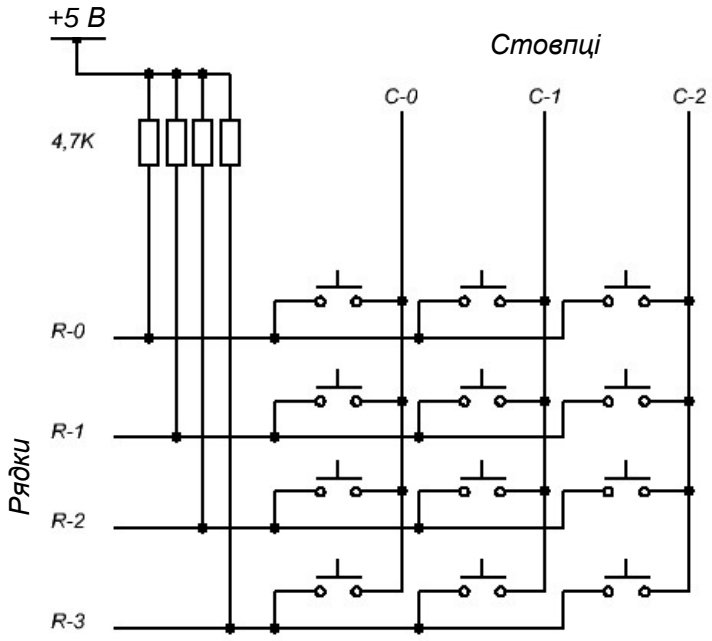


Рис.2.20. Матрична клавіатура 4×3

Клавіші розташовані у вузлах матриці, у якої чотири лінії рядків і три лінії стовпців. На лінії стовпців по черзі подається негативний імпульс (логічний «0»). У цей момент перевіряється стан чотирьох ліній рядків. Якщо натиснутих клавіш немає, всі лінії рядків мають високий рівень (вони підключені до напруги +5 В через резистори). Якщо ж клавіша натискається, і на лінії стовпця, відповідного натиснутій клавіші, все ще нуль, то адекватна

лінія рядка також стає рівною нулю. Знаючи номери стовпця і рядка, можна отримати позицію натиснутої клавіші.

У клавіатурі з кодуванням застосовують спеціалізовані мікросхеми, які виявляють натискання клавіші і передають її код. Прикладом такого пристрою є мікросхема MM74C922 (National Semiconductors) [2].

Як оптичний датчик найчастіше виступає світлодіод та фотоприймач, який називається оптроном. Випускаються оптрони з закритим (optoisolator) оптичним каналом (у монолітному виконанні) і відкритим оптичним каналом (щілинні і відбивальні оптрони).

На рис.2.21 показаний оптичний датчик. щілинний оптрон (slotted optical switch). Фототранзистор і направлений на нього світлодіод закріплені на пластиковій підставці і розділені проміжком так, що коли якийсь предмет рухається в зазорі, він перекриває світло між світлодіодом і датчиком. Щілинні оптрони часто використовуються для вимірювання швидкості двигуна за допомогою диска з прорізами, розміщеного на осі двигуна. Коли вісь обертається, диск перекриває світловий промінь. Інше застосування щілинного оптрона. це індикація того, відкриті чи закриті двері або, наприклад, кожух охоронного приладу. Прапорець на дверях, потрапляючи в щілину, блокує світло, коли двері закриваються. Механічна комп'ютерна миша також використовує щілинні оптрони.

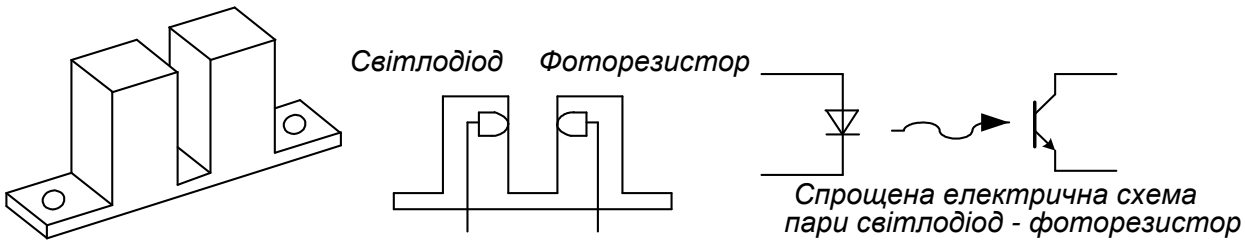


Рис.2.21. Щілинний оптрон

На рис.2.22 показаний інший тип оптичного датчика. відбивальний оптрон (reflective sensor). Принцип роботи цього датчика такий же, як і щілинного, з тією різницею, що фототранзистор приймає відбите, а не пряме

світло. Більшість датчиків відбиття характеризується фокусною відстанню. оптимальною відстанню, на якій має бути розміщений відбивальний об'єкт від датчика. Ця відстань дорівнює 0,254... 1,270 см (від 0,1 до 0,5 дюйма). Типове застосування відбивальних оптронів. це реєстрація обертання двигуна за нанесеними на його вісь темними мітками [2, 10].

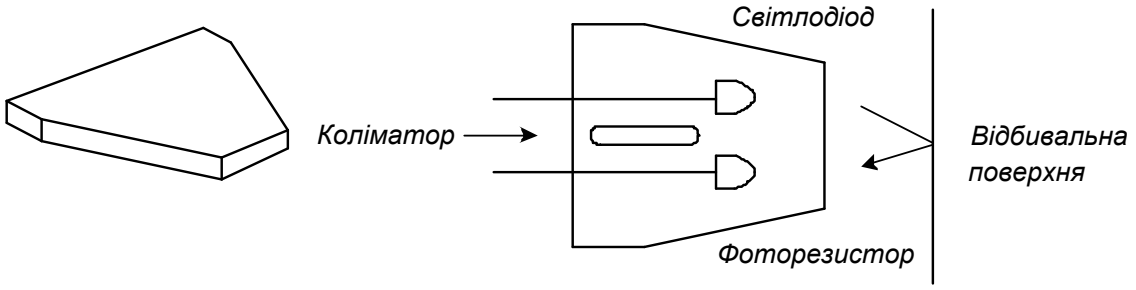


Рис.2.22. Відбивальний оптрон

Коли вісь обертається, датчик реєструє зміну темних і відбивальних ділянок. Обидва типи оптронів мають схожі характеристики, які необхідно враховувати при проектуванні систем.

2.5 Пристрої формування звукових сигналів. Пристрої управління двигунами постійного струму

П'єзоелектричні динаміки служать для генерації звуків. Вони мають максимальну вхідну напругу 50 В і номінальний струм 10 мА. На рис.2.23 зображена схема, що використовує буфер для керування таким динаміком. Схема пристрою керування на транзисторі ZTX300 наведена на рис.2.23 [2, 9]. Щоб отримати звук, необхідно подати на вхід послідовність імпульсів.

Напівпровідникові зумери. це автономні динаміки, здатні генерувати тон, частотою близько 450 Гц. На рис.2.24 наведена схема керування на транзисторі *ZTX300*. Для генерації звуку на базу *ZTX300* необхідно подати високий рівень напруги. При керуванні сиренами можна використовувати такі ж схеми.

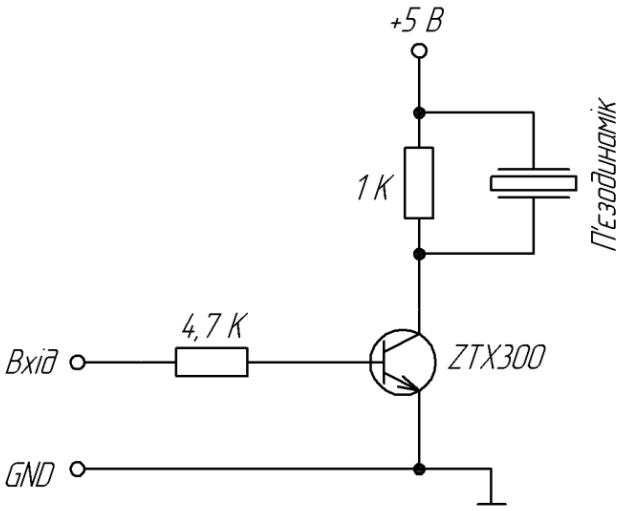


Рис.2.23. Схема керування п’єзоелектричним динаміком

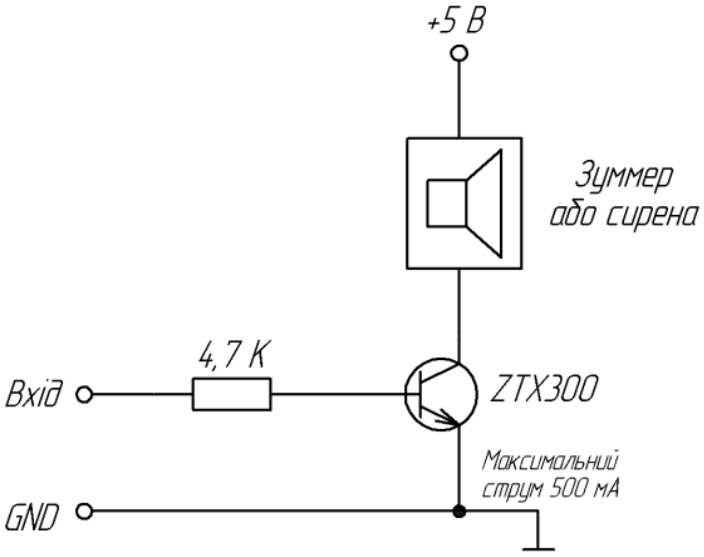


Рис.2.24. Схема керування зумером або сиреною

На рис.2.25 зображена схема формування звукового сигналу апаратним способом. Схема складається з генератора звукового діапазону на логічних елементах І та схеми керування звуковим пристроєм. При формуванні МК сигналу логічної «1» імпульси від генератора проходять на звуковий пристрій.

Двигунами постійного струму можна керувати за допомогою реле або транзисторів (рис.2.26). Одиночне перемикальне реле вмикає і вимикає двигун, а спарене реле відповідає за напрям обертання (рис.2.26).

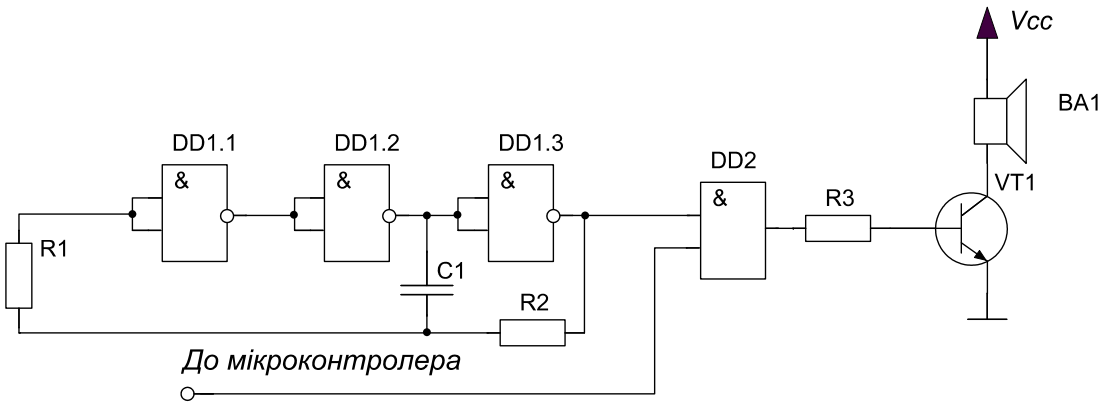


Рис.2.25. Схема формування звукового сигналу апаратним способом

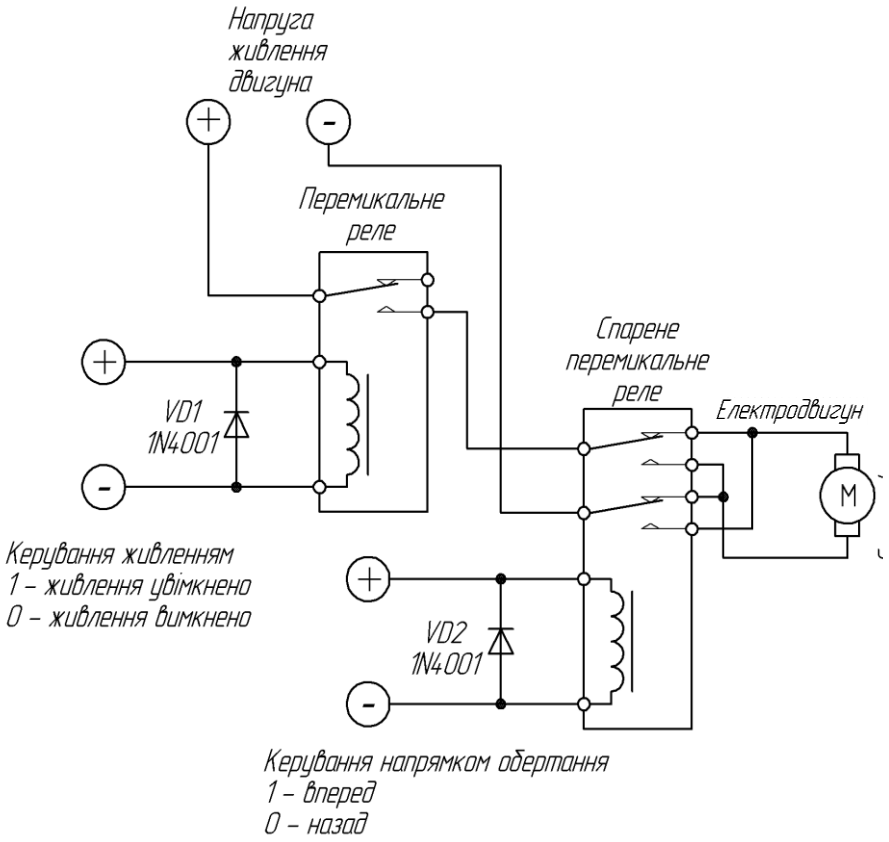


Рис.2.26. Схема керування двигуном постійного струму на базі реле

Інший спосіб керування двигунами постійного струму заснований на використанні мостових схем типу L298N (SGS-Thomson, RS636-384) [9, 10]. Це двоканальний пристрій, в якому присутня потужна напруга до 46 В, струм до 2А на кожен канал. Такий пристрій працює від рівнів ТТЛ (рис.2.27).

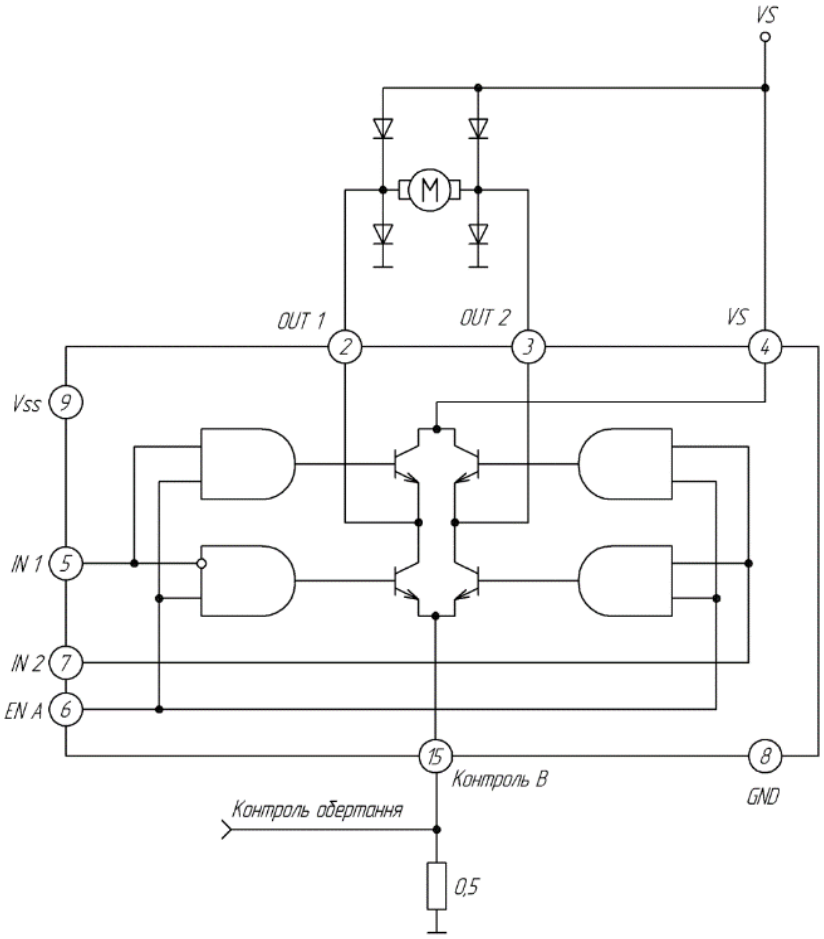


Рис.2.27. Схема керування двигуном постійного струму

З виводу VS (контакт 4) подається напруга живлення для двигуна, на вивід Vss (контакт 9) подається напруга живлення схеми (+5В). Виводи EN A і EN B (контакти 6 і 11) відкривають входи двох каналів. Входи IN 1 і IN 2 (контакти 5 і 7) керують першим каналом, а IN 3 і IN 4. другим. Емітери транзисторів з’єднані для підключення зовнішніх датчиків контролю. Типова схема включення для одного каналу наведена на рис.2.27. Коли на вході EN A низький рівень, входи заблоковані і двигун не обертається. Якщо на цей вхід подати високий рівень, входи відкриваються. Входи IN 1 і IN 2 керують режимами роботи двигуна таким чином: IN 1 «1», IN 2 «0». двигун обертається за годинниковою стрілкою; IN 1 «0», IN 2 «1». двигун обертається проти годинникової стрілки; IN 1 = IN 2. двигун не обертається.

При керуванні колекторними двигунами постійного струму потрібно регулювати струм, що проходить через обмотки двигуна. Цей процес охоплює регулювання напрямку магнітного потоку і величини струму. Проста схема керування наведена на рис.2.28. Дані схеми дозволяють керувати обертанням двигуна лише в одному напрямку.

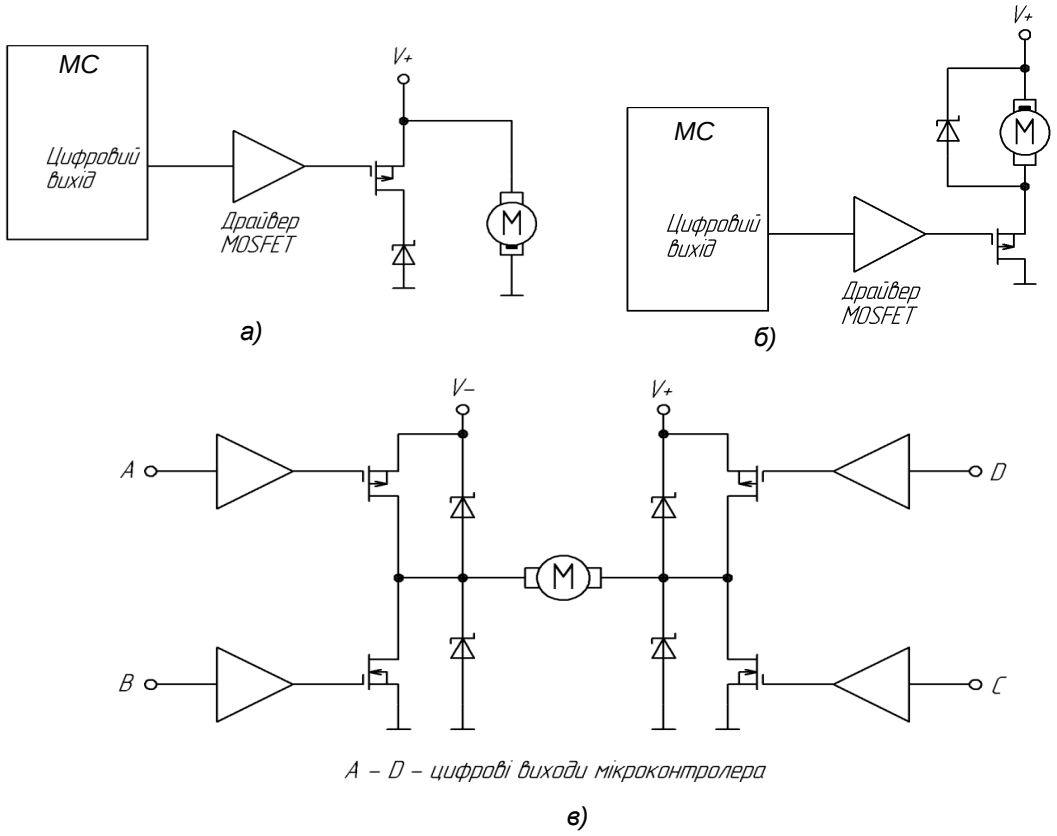


Рис.2.28. Керування колекторним двигуном постійного струму: а) схема з верхнім розташуванням ключа; б) схема з нижнім розташуванням ключа; в) мостова схема

Схема з верхнім розташуванням ключа часто застосовується в системах з підвищеними вимогами до безпеки. коротке замикання не призводить до увімкнення двигуна, у вимкненому стані обидва виводи обмотки підключено до спільної точки схеми. Схема з нижнім розташуванням ключа найдешевша, оскільки для керування силовим транзистором MOSFET досить подавати на затвор сигнал з цифрового виходу МК без використання спеціального драйвера. Для реверсивного керування двигуном потрібно використовувати

мостову схему включення, наведену на рис.2.28. Частота обертання двигуна регулюється за допомогою зміни значення напруги на обмотці якоря. При використанні МК цю напругу можна регулювати за допомогою широтно-імпульсної модуляції.

Для вимірювання частоти обертання двигуна можна використовувати ефект зворотної ЕРС або використовувати оптоелектронний датчик положення ротора («ромашка») (рис.2.29).

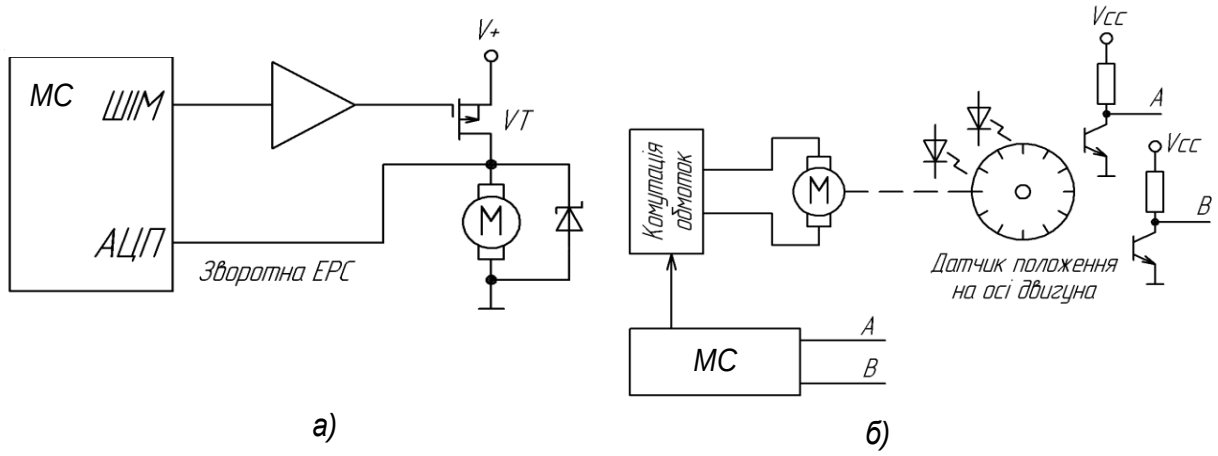


Рис.2.29. Вимірювання швидкості обертання двигуна: а) з використанням зворотної ЕРС; б) з використанням датчика положення ротора

Безколекторні двигуни є прикладом спрощення конструкції з одночасним ускладненням схеми керування. Двигун не може самостійно перемикати обмотки (керувати струмом), тому схема керування повинна самостійно правильно регулювати величину струму в обмотках для забезпечення рівномірного обертання валу двигуна. Схема керування містить півмостову схему включення кожного з трьох виводів обмоток. Існують 2 основних типи керування безколекторним двигуном: з датчиками і без датчиків. Для того, щоб включати обмотки в потрібній послідовності, необхідно використовувати різні методи визначення положення ротора. Двигун з датчиком завжди повідомляє МК про положення ротора. Кожному положенню ротора відповідає певний набір керівних дій, які подаються на мостову схему включення обмоток.

У двигунах без датчика положення ротора визначається за величиною ЕРС, що виникає в непідключеній обмотці. Двигуни без датчиків простіші у виготовленні, але складніші в управлінні. Їх застосовують в умовах, що не вимагають частих запусків і зупинок. Двигуни з датчиками. кращий вибір для умов, пов'язаних з періодичними зупинками і запусками. Схеми включення двигуна наведені на рис.2.30. Складність побудови схем керування не залежить від типу двигуна. Безколекторні двигуни мають кращі показники надійності, питомої потужності і економічності в порівнянні з колекторними, тому рекомендується використовувати безколекторні двигуни.

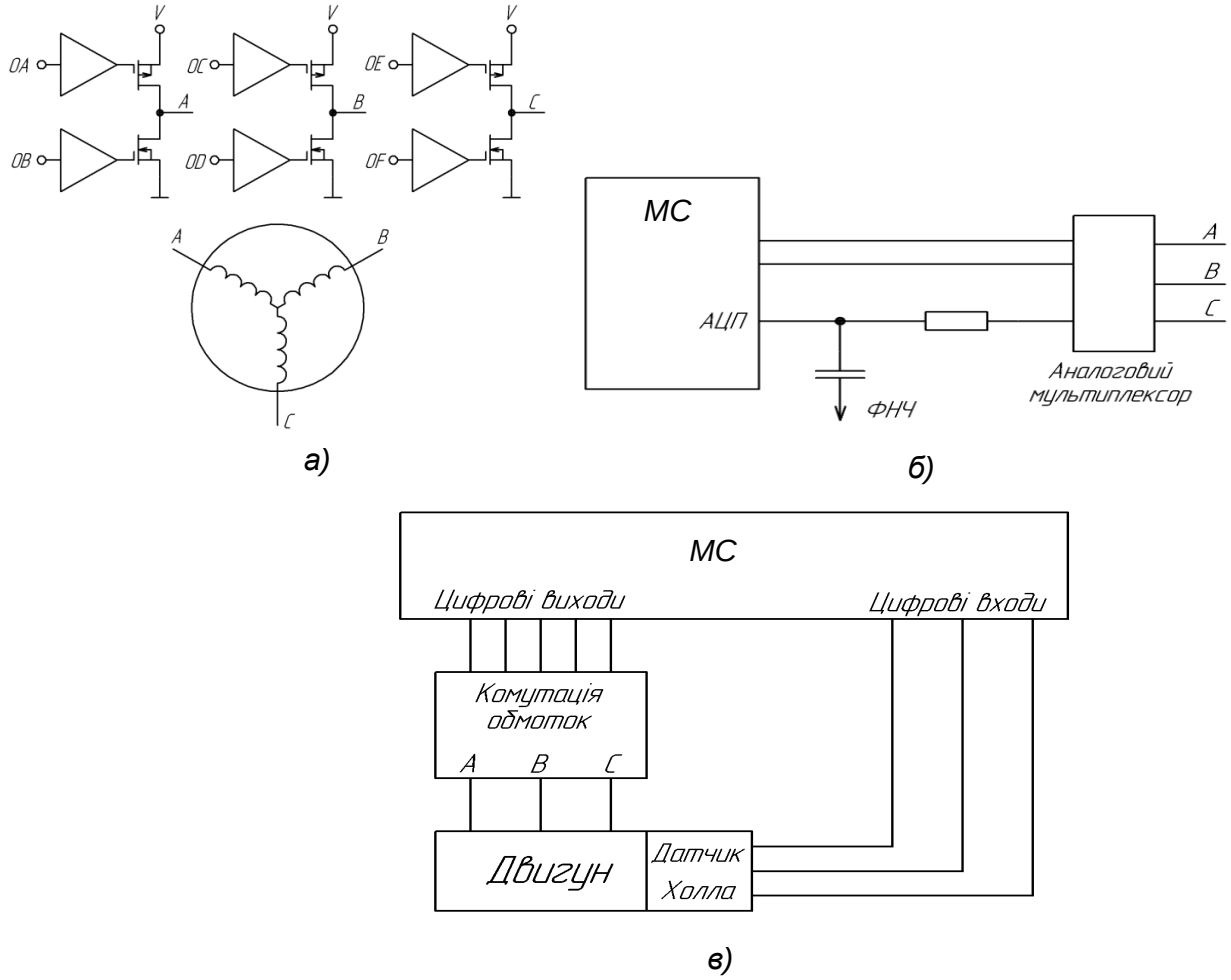


Рис.2.30. Керування безколекторним двигуном постійного струму: а) спільна схема включення; б) схема включення без датчика положення; в) схема включення з датчиком положення.

2.6 Периферійний послідовний інтерфейс UART, SPI

До складу мікроконтролерів AVR входить універсальний дуплексний послідовний порт (UART). Його основні можливості: широкий діапазон швидкостей обміну даними; висока швидкість передачі при низькій частоті XTAL; 8 або 9-розрядний формат даних; виявлення помилок утрати даних при прийомі; виявлення помилок формату кадрів; виявлення помилкового стартового біта; три окремих переривання: по завершенню передачі, по порожньому регістру передавача і по завершенню прийому.

При передачі модуль UART додає до вхідного символу (8 або 9 біт) на початку. старт-біт (нуль), а в кінці. стоп-біт (одиниця), формуючи таким чином 10- або 11-бітову послідовність. Отримані значення передаються до регістра зсуву, який по черзі передає біти на вихід передавача TXD (вивід PD1). Швидкість видачі біт на вихід передавача визначається параметром *baud rate* (швидкість передачі інформації; вимірюється в бодах), яким можна керувати.

Приймач модуля UART безперервно перевіряє стан входу RXD, на якому за відсутності даних встановлюється рівень «1». Приймач зчитує інформацію з входу в 16 разів швидше. При виявленні на виводі RXD рівня «0» (тобто можливого старт-біта) мікроконтролер пропускає шість відліків, а потім робить три вибірки. Ці вибірки доводяться на відлік 8, 9 і 10 для кожного біта, що приймається, і, таким чином, зчитування значення біта відбувається в середині інтервалу його передачі, що дозволяє працювати з сигналами, що мають фронти великої тривалості. Якщо мікроконтролер виявляє, що на виводі RXD все ще присутній рівень «0», тобто прийшов стар-біт, модуль UART переходить в робочий режим і починає зчитувати байт. Якщо ж на виводі RXD вже присутній рівень «1», вважається, що перший відлік був просто шумом, і модуль переходить до очікування коректного символу. Якщо приймач визначив, що прийшов дійсний символ, він починає брати по три відліки кожного біта в середині інтервалу його передачі. Якщо значення всіх

трьох відліків біта не збігаються, то значення біта набуває рівним значенню двох однакових відліків. На завершення модуль зчитує вибірки, що відносяться до стоп-біту. Для того, щоб було вирішено про коректний прийом символу, принаймні, дві з цих вибірок мають дорівнювати одиниці. Інакше модуль вважає символ за невірно кадрований і реєструє помилку кадрування (framing error).

Периферійний послідовний інтерфейс SPI (рис.2.31) застосовується як для з'єднання МК між собою, так і МК з периферійними пристроями. У одному сеансі зв'язку беруть участь лише 2 пристрої, з яких один обов'язково МК, а інший або МК, або периферійний пристрій з інтерфейсом SPI (АЦП, датчик, пам'ять, виконавчий пристрій). У периферійному послідовному інтерфейсі SPI використовуються цифрові сигнали «такти» (clock), «вибір кристала (мікросхеми)» (chip select), «вхід даних» (data input) і «вихід даних» (data output), але немає адресних сигналів.

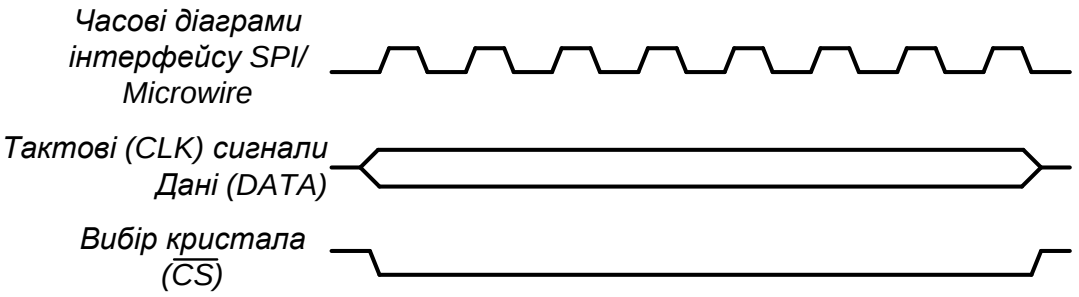


Рис.2.31. Шина SPI

Кожен з пристроїв, підключених до шини SPI, вимагає наявності окремого сигналу $\overline{CS}$, яким він вибирається. Два пристрої працюють в режимі «ведучий-ведений». Дані, які потрібно передати, завантажуються в 8-бітові регістри портів і вхід даних ведучого пристрою з'єднується з виходом даних веденого. У свою чергу, вихід даних ведучого пристрою з'єднується з входом даних веденого. Також підключені до МК (ведучого пристрою) останні вище перелічені ведені пристрої, які включаються за окремими сигналами «вибір кристала». Хоча регістри мають по 8 бітів, але при такому підключенні входів і виходів

утворюється спільний 16-бітовий регістр зсуву і, крім того, одночасно передаються по двох лініях дані в обох напрямах. Зсув проводиться тактовими сигналами від тактового генератора ведучого пристрою.

2.7 Організація обміну даними інтерфейсом I^2C, 1-Wire

У двонаправленій шині I^2C використовується лише 2 лінії: послідовна лінія синхронізації SCL (SCLock) і послідовна лінія даних SDA (SDAta) [2]. Сигнал на лінії SCL формується процесором для синхронізації даних периферійного пристрою. Обидва виводи (SDA і SCL) виконуються з відкритим колектором або відкритим стоком, що визначається типом мікросхеми. Вони з'єднані з позитивним джерелом живлення через резистор навантаження за схемою «Монтажне І» і, таким чином, декілька пристроїв можуть одночасно використовувати шини SCL і SDA.

При передачі даних сигнал SDA можна змінювати лише доти, доки на SCL встановлений низький рівень. Коли на SCL високий рівень, перепади на лінії SDA з одного рівня в інший інтерпретуються як умови «СТАРТ» і «СТОП». Якщо SDA переходить у низький рівень, тоді як на лінії SCL високий, усі периферійні пристрої на шині сприйматимуть цю подію як старт-умову. Якщо SDA переходить у високий рівень, коли на SCL. високий, генерується стоп-умова. Процесор генерує старт-умову, а потім посилає одночасно всім периферійним пристроям адресу довжиною 7 бітів, повідомляючи, який з них вибраний, і восьмий біт читання/запису (0. запис, 1. читання). 7-бітова адреса дозволяє підключати по I^2C-шині 2^7, або 128, периферійних пристроїв за однієї умови: ємність шини не повинна перевищувати 400 пФ.

Під час передачі в першому байті восьмого біта читання/запису (R/W) процесор встановлює напрям передачі: програмує R/W на запис (передачу) даних від МК до периферійного пристрою, якщо цей біт дорівнює 0. Інакше МК налаштовується на читання (прийом) даних від периферійного пристрою з адресою, вказаною в 7 бітах. Слід відмітити, що перші 7 бітів передаються,

починаючи з старшого біта і закінчуючи молодшим бітом, а восьмим бітом передається біт R/W. Після прийому кожного байту, включаючи адресний, вибраний периферійний пристрій (приймальна сторона) по лінії SDA посилає сигнал підтвердження на виконання функції прийому переведенням рівня на лінії SDA в низький, щоб показати, що він прийняв адресу і умову читання/запису.

Після прийому біта підтвердження процесор встановлює адресу іншого периферійного пристрою, з яким він хоче встановити зв'язок. Довжина цього поля залежить від периферійного пристрою. Потім приймається біт підтвердження і передаються дані. При виконанні операції запису процесор синхронізує вихідні 8 бітів, а при читанні МП встановлює виведення SDA як вхід і синхронізує вхідну 8-бітову послідовність. Дані завершуються бітом підтвердження.

Деякі периферійні пристрої дозволяють зчитувати або записувати декілька байтів за одну передачу. Процесор повторює послідовність команд «Дані/біт підтвердження» (data/acknowledge) до тих пір, поки всі байти не будуть передані. Периферійний пристрій збільшуватиме свою внутрішню адресу після кожної передачі.

Шина 1-Wire є основою мереж MicroLAN і розроблена наприкінці XX століття фірмою Dallas Semiconductor [2, 9]. Шина 1-Wire побудована за технологією Master/Slave. На шині повинен бути хоча б один ведучий пристрій (Master). Всі інші пристрої повинні бути веденими (Slave). Ведучий пристрій ініціює всі процеси передачі інформації в межах шини. Master може прочитати дані з будь-якого Slave-пристрою або записати їх туди. Передача інформації від одного Slave-пристрою до іншого безпосередньо неможлива. Для того, щоб Master міг звертатися до кожного з ведених пристроїв по шині, кожний ведений пристрій містить у собі індивідуальний код (ID-код).

Протокол l-Wire містить у собі спеціальну команду пошуку, за допомогою якої ведучий пристрій (Master) може здійснювати автоматичний пошук ведених пристроїв. У процесі пошуку Master визначає ID-коди для всіх

підключених до мережі мікросхем. Пошук відбувається шляхом поступового відсіювання неіснуючих адрес. Тому для того, щоб знайти всі пристрої, що підключені до шини, потрібен досить значний час. Середня швидкість пошуку елементів у мережі MicroLAN становить близько 75 вузлів за секунду.

Обмін інформацією по шині 1-Wire відбувається за такими правилами.

1. Обмін завжди ведеться з ініціативи одного ведучого пристрою, що у більшості випадків є мікроконтролером.

2. Будь-який обмін інформацією починається з подачі імпульсу скидання («Reset Pulse» або просто RESET) у лінію 1-Wire ведучим пристроєм.

3. Для інтерфейсу 1-Wire у загальному випадку передбачається «гаряче» підключення й відключення пристроїв.

4. Будь-який пристрій, підключений до 1-Wire, після одержання живлення видає в лінію DQ імпульс присутності, який називається «Presence pulse». Цей же імпульс пристрій завжди видає в лінію, якщо виявить сигнал RESET.

5. Поява в шині 1-Wire імпульсу PRESENCE після видачі RESET однозначно інформує про наявність хоча б одного підключеного пристрою.

6. Обмін інформації ведеться так званими тайм-слотами: один тайм-слот служить для обміну одним бітом інформації.

7. Дані передаються побайтово, біт за бітом, починаючи з молодшого біта. Ймовірність переданих/прийнятих даних (перевірка відсутності спотворень) гарантується шляхом підрахунку циклічної контрольної суми.

На рис.2.32 показана діаграма сигналів RESET і PRESENCE, з яких завжди починається будь-який обмін даними.

Імпульс RESET формує МК, що переводить в низький логічний рівень шину 1-Wire і втримує її в цьому стані мінімум 480 мікросекунд. Далі МК повинний «відпустити» шину. Через якийсь час, що залежить від ємності лінії й опору резистора, що підтягує, у лінії встановиться високий логічний рівень. Протокол 1-Wire обмежує цей час «релаксації» діапазоном від 15 до 60 мікросекунд, що і є визначальним для вибору резистора, що підтягує.

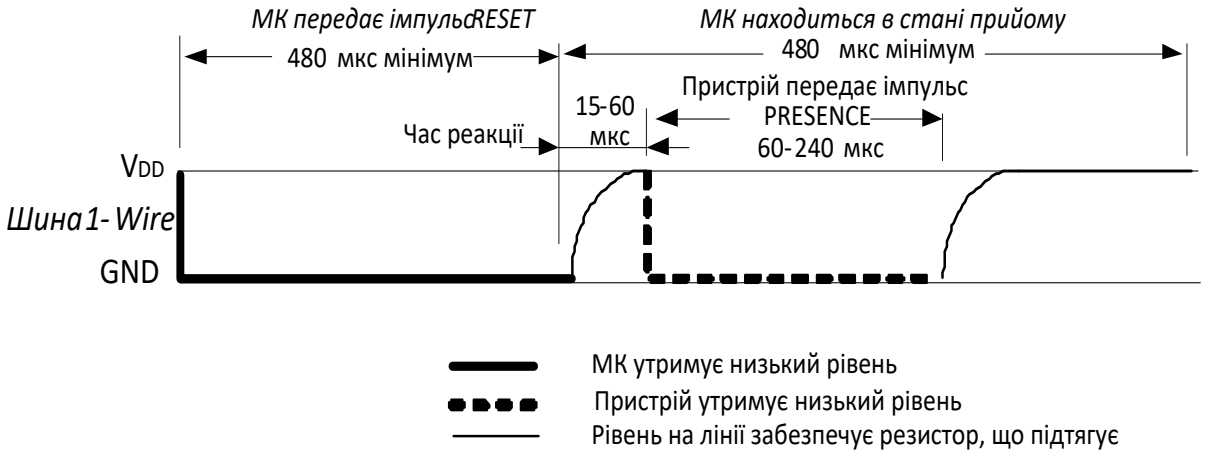


Рис.2.32. Діаграма сигналів RESET і PRESENCE

Виявивши імпульс RESET, ведений пристрій (Slave) приводить свої внутрішні вузли у вихідний стан і формує відповідний імпульс PRESENCE не пізніше 60 мікросекунд після завершення імпульсу RESET. Для цього пристрій переводить у низький рівень лінію і втримує її в цьому стані від 60 до 240 мікросекунд. Після цього пристрій так само «відпускає» шину.

Але після завершення імпульсу PRESENCE пристрою дається ще якийсь час для завершення внутрішніх процедур ініціалізації, таким чином, МК повинен приступити до будь-якого обміну з пристроєм не раніше, ніж через 480 мікросекунд після завершення імпульсу RESET.

Отже, процедура ініціалізації інтерфейсу, з якої починається будь-який обмін даними між пристроями, триває мінімум 960 мікросекунд, складається з передачі від МК сигналу RESET і прийому від пристрою сигналу PRESENCE. Якщо сигнал PRESENCE не виявлений. значить на шині 1-Wire немає готових до обміну пристроїв.

Обмін бітами інформації здійснюється певними тайм-слотами. Тайм-слот. це певна, досить жорстко лімітована за часом послідовність зміни рівнів сигналу в лінії 1-Wire. Розрізняють 4 типи тайм-слотів: передача «1» від МК, передача «0» від МК, прийом «1» від Slave-пристрою, прийом «0» від Slave-пристрою.

Будь-який тайм-слот завжди починає МК шляхом переведення шини 1-Wire у низький логічний рівень. Тривалість будь-якого тайм-слоту повинна перебувати в межах від 60 до 120 мікросекунд. Між окремими тайм-слотами завжди повинен передбачатися інтервал не менший 1 мікросекунди.

Тайм-слоти передачі відрізняються від тайм-слотів прийому поводженням МК: при передачі він тільки формує сигнали; при прийманні, крім того, ще й опитує рівень сигналу в лінії 1-Wire. рис.2.33 демонструє часові діаграми тайм-слотів всіх 4-х типів: угорі показані тайм-слоти передачі від МК, унизу - прийому від Slave-пристрою.

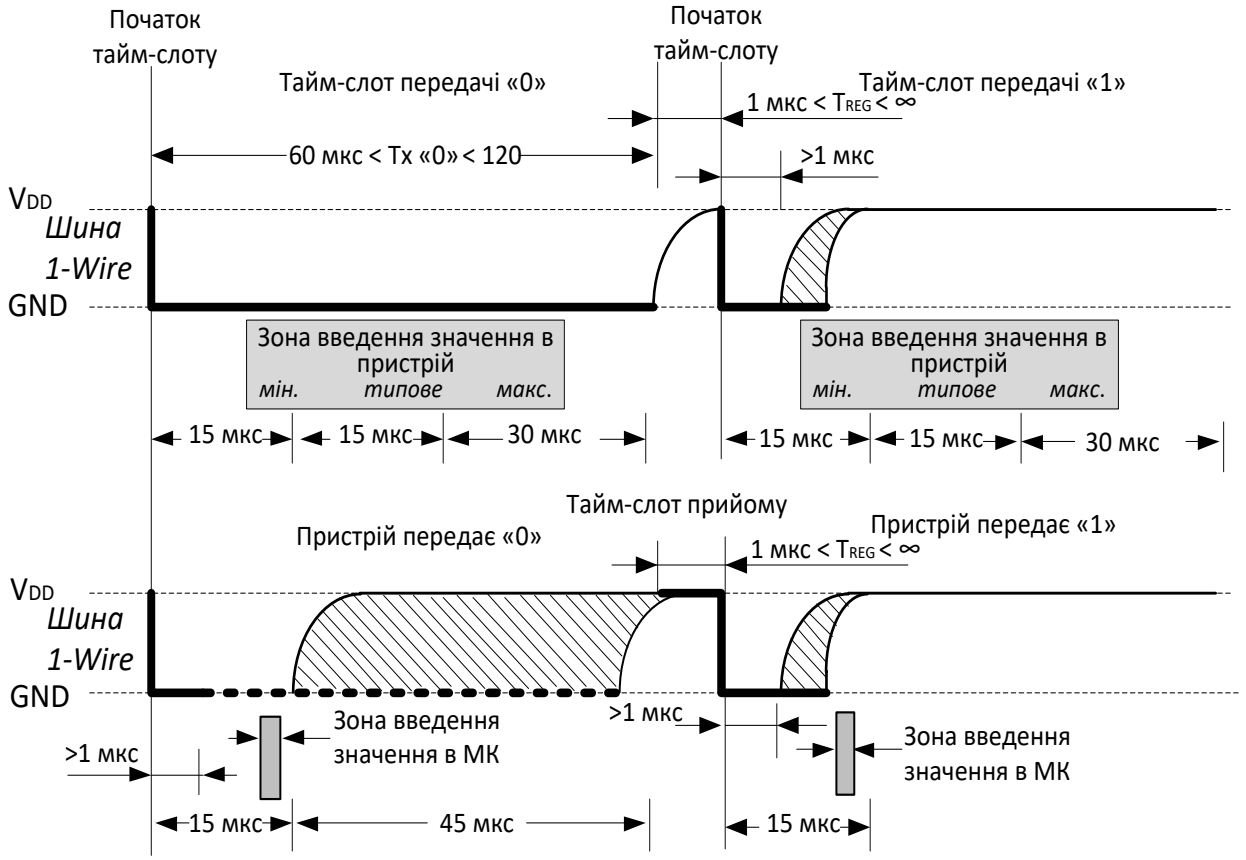


Рис.2.33. Часові діаграми тайм-слотів на лінії 1-Wire

Кожний пристрій 1-Wire має унікальний ідентифікаційний 64-бітовий номер, який програмується на етапі виробництва мікросхеми.

На мережевому рівні відбувається адресація елементів на однопроводовій шині. Перебуваючи на цьому рівні, всі Slave пристрої очікують одну з команд

адресації. Така команда повинна вибрати один або кілька пристроїв, з якими Master буде працювати на транспортному рівні.

Після того, як МК видасть команду READ ROM (0×33), від пристрою надійде 8 байтів його власної унікальної адреси. МК повинен їх прийняти. Будь-яка процедура обміну даними з пристроєм повинна бути завершена повністю або перервана посиланням сигналу RESET.

Якщо відправлено команду MATCH ROM (0×55), то після неї МК повинен передати так само й 8 байтів конкретної адреси пристрою, з яким буде здійснюватися наступний обмін даними. За цією командою кожний пристрій порівнює передану адресу зі своєю власною. Всі пристрої, адреси яких не збіглися, припиняють аналіз і видачу сигналів у лінії 1-Wire, а пристрій, що пізнав адресу, продовжує роботу. Тепер всі дані, що передані МК, будуть потрапляти лише до цього пристрою. Які саме дані треба послати в пристрій або одержати від нього після його адресації, залежить від конкретного пристрою.

Якщо пристрій один на шині, то можна прискорити процес взаємодії з ним за допомогою команди SKI ROM (0×CC). Отримавши цю команду, пристрій відразу вважає, що адреса збіглася, хоча ніякої адреси для цієї команди не треба. Деякі процедури не вимагають прийому від пристрою ніяких даних, у цьому випадку команду SKI ROM можна використовувати для передачі деякої інформації відразу всім пристроям.

Всі мікросхеми мережі MicroLAN переходять на транспортний рівень після будь-якої команди мережного рівня. Набір команд транспортного рівня для різних мікросхем дещо відрізняється. Основні команди транспортного рівня. це команди «Запис пам'яті» і «Читання пам'яті».

Структура індивідуального коду, записаного в спеціальний ПЗП будь-якої мікросхеми, що підтримує інтерфейс 1-Wire, умовно зображена на рис.2.34. Код складається з трьох основних елементів:

Рис.2.34. Структура індивідуального коду мікросхеми 1 -Wire

Перші 8 бітів, починаючи з молодшого розряду,. це груповий код мікросхеми. Груповий код однозначно визначає її вид. Наступні 48 бітів. це унікальний серійний номер мікросхеми. Останні 8 бітів. це контрольна сума (CRC).

Контрольна сума або CRC. це байт, значення якого передається найостаннішим і обчислюється за спеціальним алгоритмом на основі значення усіх семи попередніх байтів. Алгоритм підрахунку такий, що якщо усі байти, що передаються-приймаються, без спотворень, то прийнятий байт контрольної суми співпадає з розрахованим в МК значенням. Тобто при реалізації програмного алгоритму обміну інформацією необхідно при передачі і прийомі байтів підраховувати їх контрольну суму за певним алгоритмом, а потім або передати отримане значення, або порівняти розрахункове значення з отриманим значенням CRC. Тільки при збігу обох CRC МК вважає прийняті дані достовірними. Інакше продовження обміну неможливе.

2.8 Організація обміну між ПК та МК по інтерфейсу USB

Для зв'язку з платою Arduino можна використовувати спеціальну програму моніторингу послідовного порту (Serial Monitor), вбудовану в програмне забезпечення Arduino. Монітор послідовної шини відображає дані, що посилаються в плату Arduino через віртуальний послідовний порт за допомогою USB. Для відправки даних вибирається швидкість передачі зі

списку, відповідна значенню Serial.begin в скетчі. Потім необхідно ввести текст і натиснути кнопку Send або Enter.

Плата Arduino Nano має один послідовний порт (також відомий як UART або USART): Serial, що використовується для зв'язку з комп'ютером через USB. Він пов'язаний з цифровими виводами 0 (RX) і 1 (TX). Таким чином, під час роботи послідовного порту, порти D0 та D1 не можуть використовуватися як цифрові входи або виходи.

Для забезпечення зв'язку плати Arduino з комп'ютером або іншими пристроями використовується клас Serial. Клас Serial містить близько 20 функцій. Функції для роботи з послідовним портом плати Arduino:

if (Serial):

- параметри. немає;
- значення, що повертаються . boolean: повертає true, якщо вказаний послідовний порт готовий до роботи;
- опис: дозволяє перевірити готовність послідовного порту.

Serial.begin(speed). Параметри:

- speed: швидкість в бітах на секунду (бодах);
- опис: задає швидкість передачі даних по послідовному інтерфейсу в бітах в секунду (бодах). Для взаємодії з комп'ютером слід використовувати одну з попередньо встановлених швидкостей обміну: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 або 115200.

Serial.end ():

- параметри. немає;
- значення, що повертаються. немає.
- опис: функція розриває послідовний зв'язок, після чого виводи Rx і Tx знову можна використовувати як виводи загального призначення. Для відновлення послідовного з'єднання необхідно використовувати функцію *Serial.begin ()*.

Serial.available():

– параметри. немає;
– значення, що повертаються: кількість байт, доступних для зчитування;
– опис: повертає кількість байт (символів) доступних для зчитування з буфера послідовного порту. Під символами розуміються дані, які вже прийняті і зберігаються в послідовному приймальному буфері (який може зберігати максимум 64 байта).

Serial.read():

– параметри. немає;
– значення, що повертаються: перший байт прийнятих даних (або -1, якщо таких нема);
– опис: зчитує дані, що надходять по послідовному інтерфейсу.

Serial.print(val) / Serial.print(val, format). Параметри:

– val: значення, яке необхідно вивести (будь-який тип даних);
– format: визначає систему числення (для цілочисельних типів), а також кількість десяткових знаків після коми (для чисел з плаваючою точкою). BIN (двійкова система), OCT (восьмерична система), DEC (десяткова система), HEX (шістнадцятирічна система). Для числа з плаваючою точкою цей параметр визначає кількість десяткових знаків після коми;
– значення, що повертаються: кількість виведених байт. Зчитування цього значення не є обов'язковим;
– опис: функція виводить через послідовний порт заданий ASCII текст у вигляді, зрозумілому для людини. При виведенні числа кожній його цифрі відповідає один ASCII-символ. Дробові числа теж виводяться у вигляді ASCII-цифр, при цьому після коми за замовчуванням залишається два десяткових знака. Байти виводяться у вигляді окремих символів, а символи та рядки виводяться без змін. «як є».

Приклад:

```
Serial.print("Hello world.") - виведе "Hello world."
Serial.print('N') - виведе "N"
```

```
Serial.print(78) - виведе "78"
Serial.print (78, BIN) - виведе "1001110"
Serial.print (78, OCT) - виведе "116"
Serial.print (78, DEC) - виведе "78"
Serial.print (78, HEX) - виведе "4E"
Serial.print(1.23456) - виведе "1.23"
Serial.println (1.23456, 0) - виведе "1"
Serial.println (1.23456, 2) - виведе "1.23"
Serial.println (1.23456, 4) - виведе "1.2346"
```

Serial.println (val) Serial.println (val, format). Параметри:

- val: значення, яке необхідно вивести (будь-який тип даних);
- format: визначає систему числення (для цілочисельних типів), а також кількість десяткових знаків після коми (для чисел з плаваючою точкою);
- значення, що повертаються: кількість виведених байт;
- опис: виводить через послідовний порт ASCII-текст в зрозумілому для людини вигляді з символами повернення каретки (ASCII 13 або '\ r') і нового рядка (ASCII 10 або 'n'). Ця команда має такі ж форми, як і *Serial.print ()*.

Приклад коду для роботи з послідовним портом

```
int incomingByte = 0; void setup() {
    Serial.begin(9600);
}
void loop() {
if (Serial.available() > 0) { incomingByte = Serial.read();
    Serial.print("I                received:                ");
    Serial.println(incomingByte, DEC);
    }
}
```

РОЗДІЛ 3. ПРАКТИКУМ ПРОГРАМУВАННЯ ARDUINO

Лабораторний макет включає в себе 6 світлодіодів, 1 світлодіод RGB, 1 потенціометр, 1 датчик температури LM35, 1 датчик температури та вологості DHT11, 4 тактових кнопки, 4 позиційний семисегментний дисплей, 1 матричний дисплей 8×8 з регістром керування MAX7219, 1 регістр зсуву 74HC595, 1 зумер, 1 LCD-дисплей 16×2, 1 годинник реального часу DS1307, універсальний роз'єм 3 Grove.

Лабораторний макет дозволяє вивчити такі теми, використовуючи Arduino:

Цифровий вихід:

- керування декількома світлодіодами;
- створення тону за допомогою зумера.

Цифровий вхід:

- обробка тактових кнопок;
- зчитування даних з датчика DHT11.

Аналоговий вхід:

- зчитування аналогових даних з потенціометра;
- зчитування даних з датчика температури LM35.

Аналоговий вихід (за допомогою ШІМ):

- генерування декількох кольорів за допомогою світлодіода RGB;
- регулювання яскравості світлодіода.

Комунікація SPI:

- регістр зсуву 74HC595;
- взаємозв’язок MAX7219 з Arduino для управління матричним дисплеєм матриці або семисегментним дисплеєм, використовуючи лише 3 порта Arduino.

Зв'язок I2C:

- зчитування, відображення та встановлення дати та часу з годинника реального часу DS1307.

Зв'язок UART:

– взаємозв’язок GROVE GPS та модуль Bluetooth з Arduino.

Інтерфейс дисплея:

– керування LCD-дисплеєм 16 × 2 символів за допомогою Arduino.

Мультиплексування:

– керування семисегментним дисплеєм, використовуючи мінімальну кількість потів Arduino.

Схема лабораторного макету наведена у додатку А. Розглянемо особливості функціональних вузлів макету.

У лабораторному макеті встановлено 6 одиничних світлодіодних індикаторів HL1-HL6 (рис.3.1) та один RGB світлодіод HL7. Елементи індикації HL1-HL7 підключені за схемою з загальним катодом (рівень HIGH включає світлодіод, LOW. виключає) .

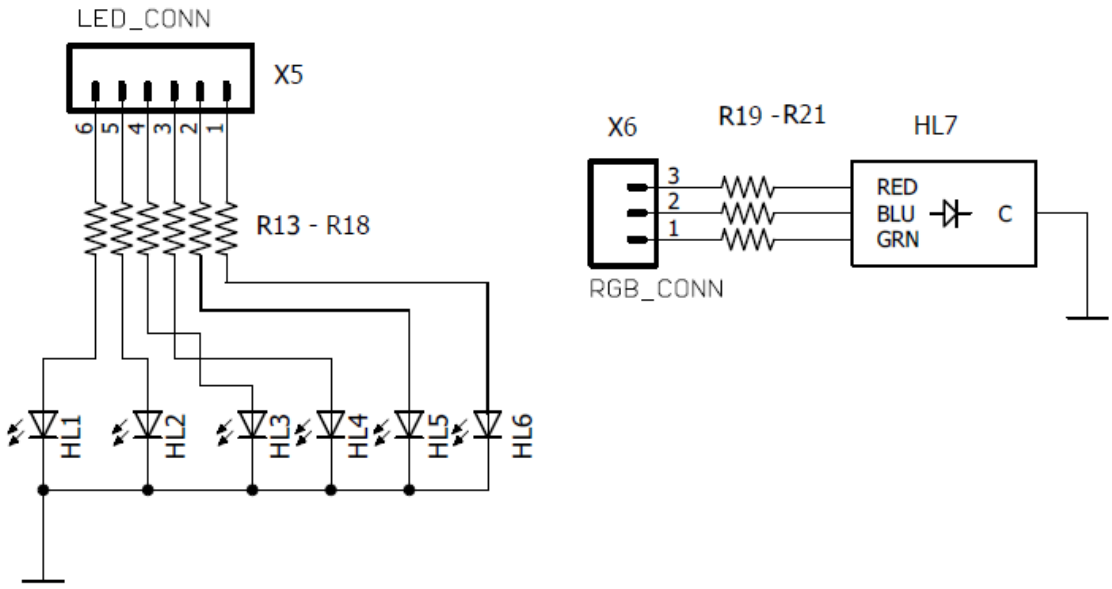


Рис.3.1. Схема світлодіодних індикаторів

Схема 4 розрядного семисегментного LED-індикатора наведена на рис.3.2. Чотирьох розрядний семисегментний індикатор має 12 контактів. 8 контактів призначені для 8 світлодіодів на кожному з семисегментних індикаторів; інші 4 контакти призначені для вибору розряду індикатора.

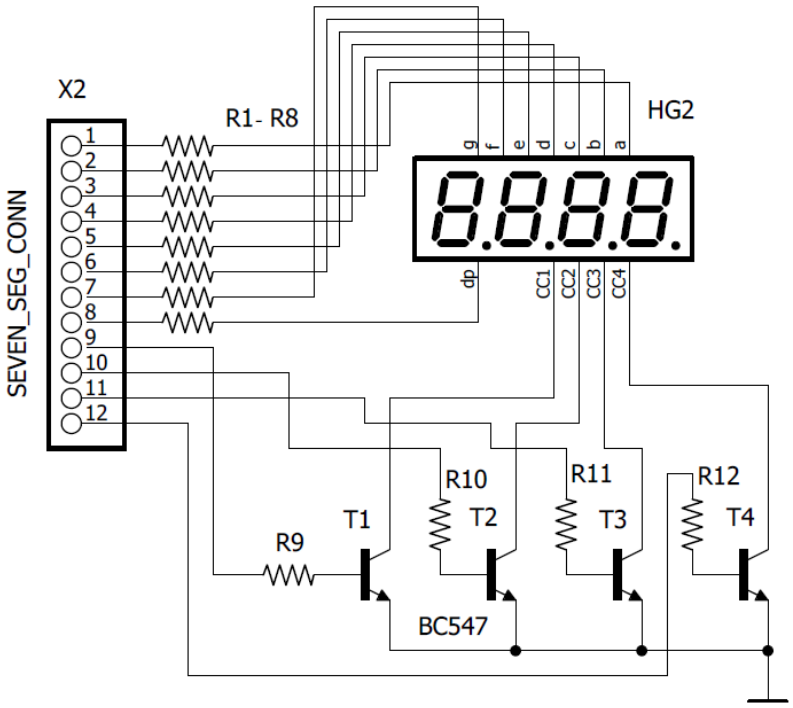


Рис.3.2. Схема 4 розрядного семисегментного LED-індикатора

Кожен сегмент модуля відображення мультиплексований, тобто він має однакові точки з'єднання анода. І кожна з чотирьох цифр модуля має власну загальну точку з'єднання катода. Це дозволяє самостійно включати або вимикати кожну цифру. Транзистори T1-T4 (BC547) використовуються як комутатори. Транзистор увімкнений, коли на базу подається позитивна напруга. Для обмеження струму транзистора використовуються резистори R9-R12 (4,7 кОм).

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.

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).

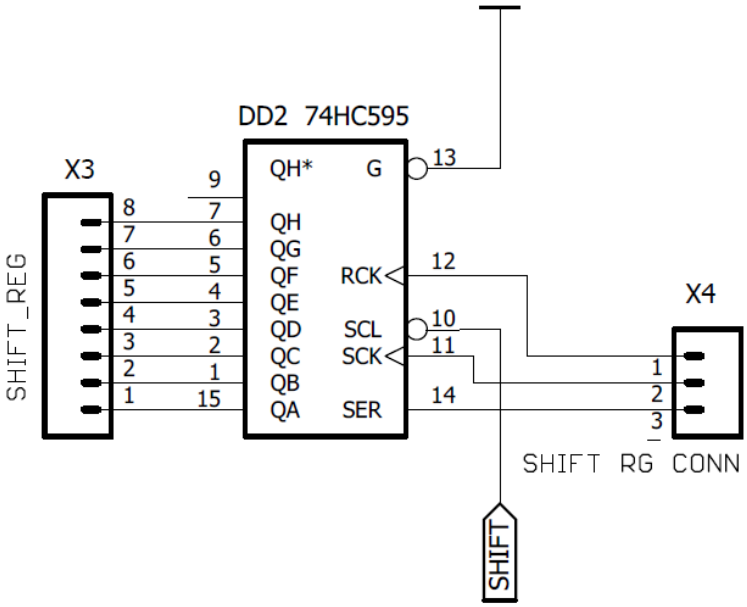


Рис.3.3. Схема модуля регістра зсуву 74HC595

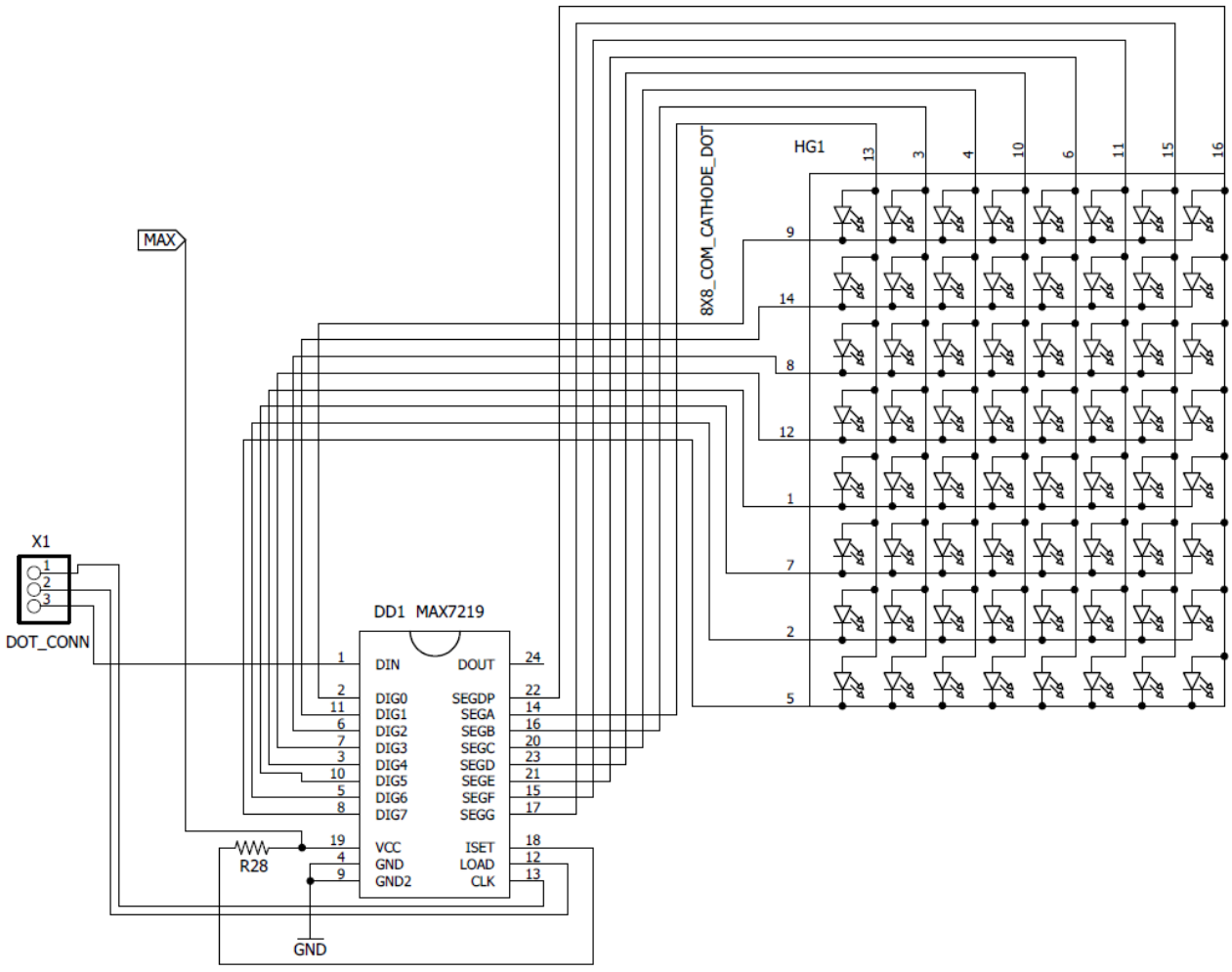


Рис.3.4. Схема модуля керування світлодіодною матриця 8×8

Світлодіодна матриця 8×8 точок (64 світлодіоди) використовується для представлення символів та зображень. 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. Живлення на мікросхему MAX7219 подається через перемикач SW1 (контакт 4 перевести у положення «On»). Виводи DIN, CLK і CS через X1 з'єднують з цифровими портами плати Arduino.

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.

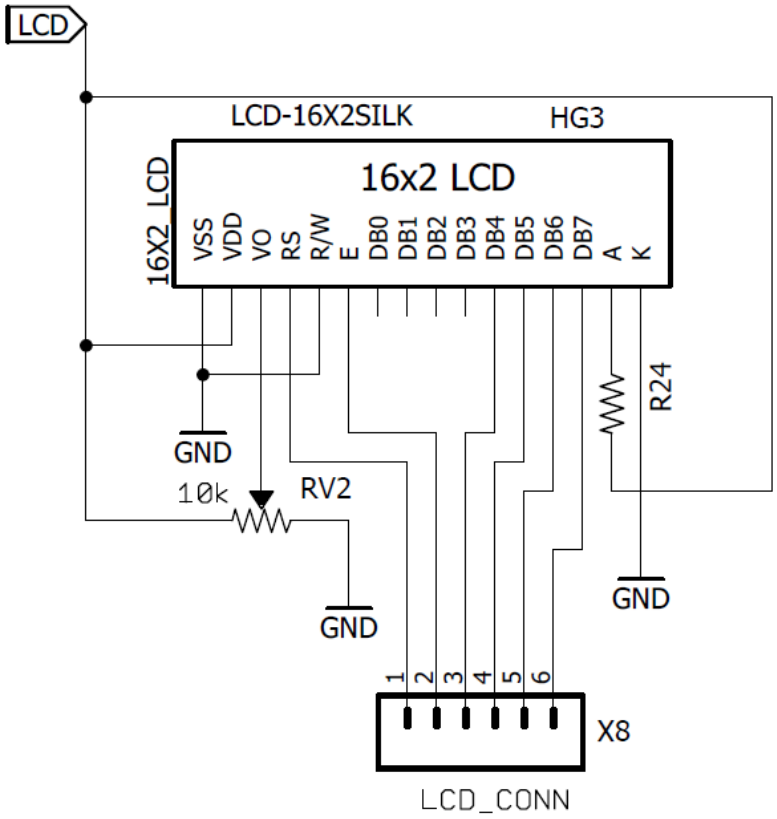


Рис.3.5. Схема модуля LCD індикатора

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»).

Схема цифрового годинника в режимі реального часу на DS1307 наведена на рис.3.6.

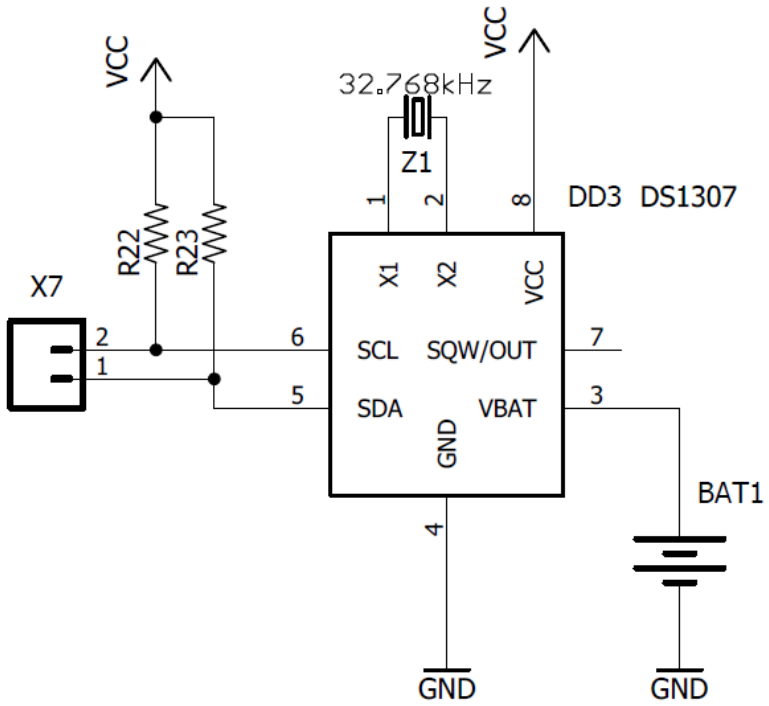


Рис.3.6. Схема модуля годинника реального часу на DS1307

DS1307. це восьми бітна мікросхема для годинника в режимі реального часу, яка використовується для підрахунку секунд, хвилин, годин, днів,

місяців та років. Для зчитування значення часу та дати з DS1307 використовується протокол зв'язку I2C. Виводи X1, X2 DD3 використовується для підключення кварцового резонатора з частотою 32,768 кГц. Вивід VBAT використовується для підключення резервного акумулятора 3В. Виводи SDA (послідовні дані/адреса) та SCL (тактові імпульси) використовується для зв'язку з іншими пристроями за допомогою протоколу зв'язку I2C. Виводи SDA та SCL потребують зовнішнього резистора підтяжки (R22, R23).

На рис.3.7 наведена схема розміщення елементів лабораторного макету «Arduino Learner Kit». Контакти DHT та TEMP_CONN призначені для зчитування інформації з датчика DHT11та LM35 відповідно. Контакт POT використовується для зняття аналогової напруги в діапазоні від 0 до 5В з змінного резистора RV1. Для підключення зумера B1 до модуля Arduino використовується контакт BUZ_CONN.

У програмному середовищі Proteus 8 віртуальний стенд реалізований у вигляді схеми, що складається з 2 аркушів [1]. На першій аркуш (рис.3.8) винесені усі основні елементи лабораторного макету. Це дозволяє оперативно аналізувати роботу лабораторного макету та програмного забезпечення. Більш детально дізнатись про роботу з програмним середовищем Proteus VSM можна [1-4].

Позначення елементів, виводи та їх адресація відповідає реальній схемі лабораторного макету (додаток А). Розташування елементів віртуального стенду, з’єднувачі та їх контакти максимально відповідають рис.3.7. Для підключення елементів до контактів Arduino Nano використовуються Terminals з’єднувачі, для яких потрібно вказати мітку (Label) з’єднання. Наприклад, для підключення контакту HL7 до порту D5 Arduino Nano, необхідно активувати режим Wire Label Mode (піктограма «LBL»), виділити Terminal та вибрати у контекстному меню режим «Edit Terminal Label». У спадному списку String вибираємо мітку D5 (рис.3.9). У реальному макеті використовується провідник для з’єднання контакту 3 (X6) та контакту 8 (X9).

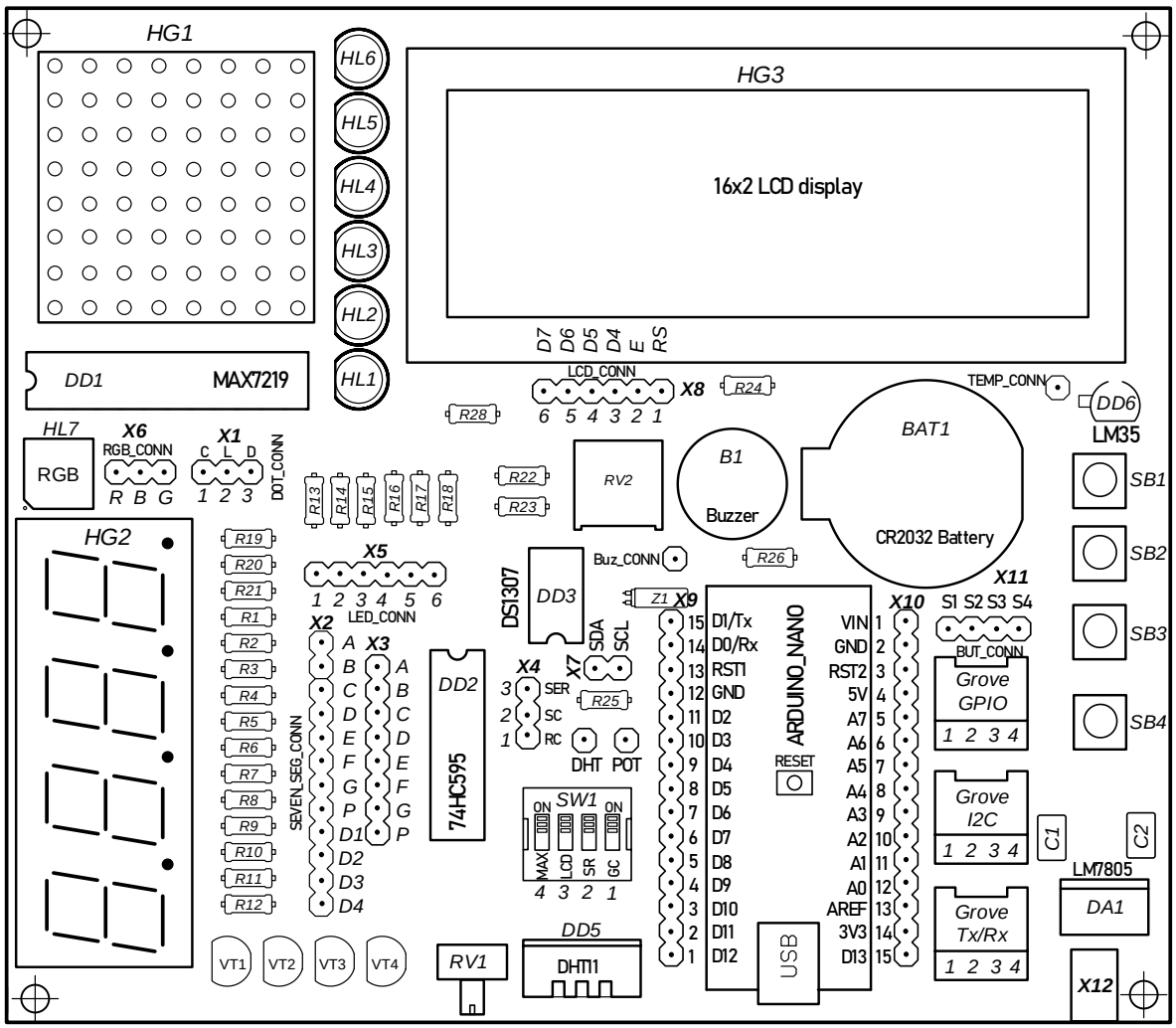


Рис.3.7. Схема розміщення елементів лабораторного макету

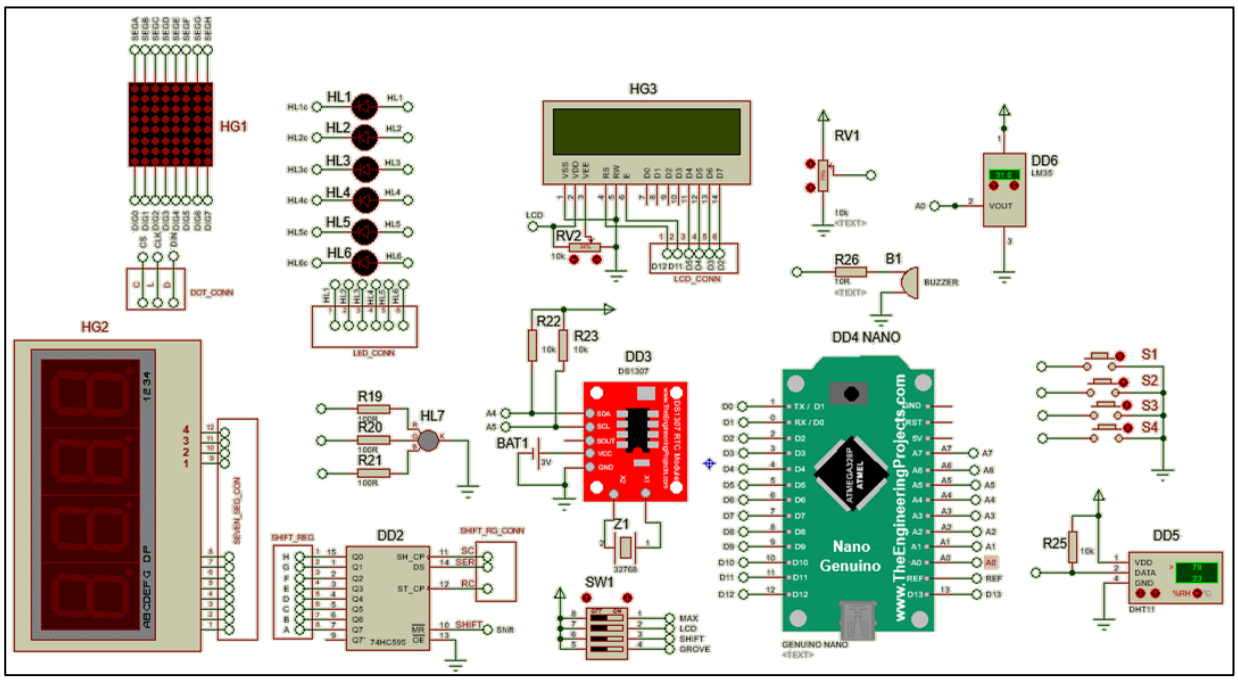


Рис.3.8. Віртуальний стенд «Arduino Learner Kit» у середовищі Proteus 8

Для завантаження HEX-файлу в мікроконтролер необхідно відкрити вікно властивостей мікроконтролера, де у полі Program File вказати шлях до файлу, як показано на рис.3.10. Файл з розширенням .HEX створюється після компіляції файлу з лістингом програми у середовищі розробки.

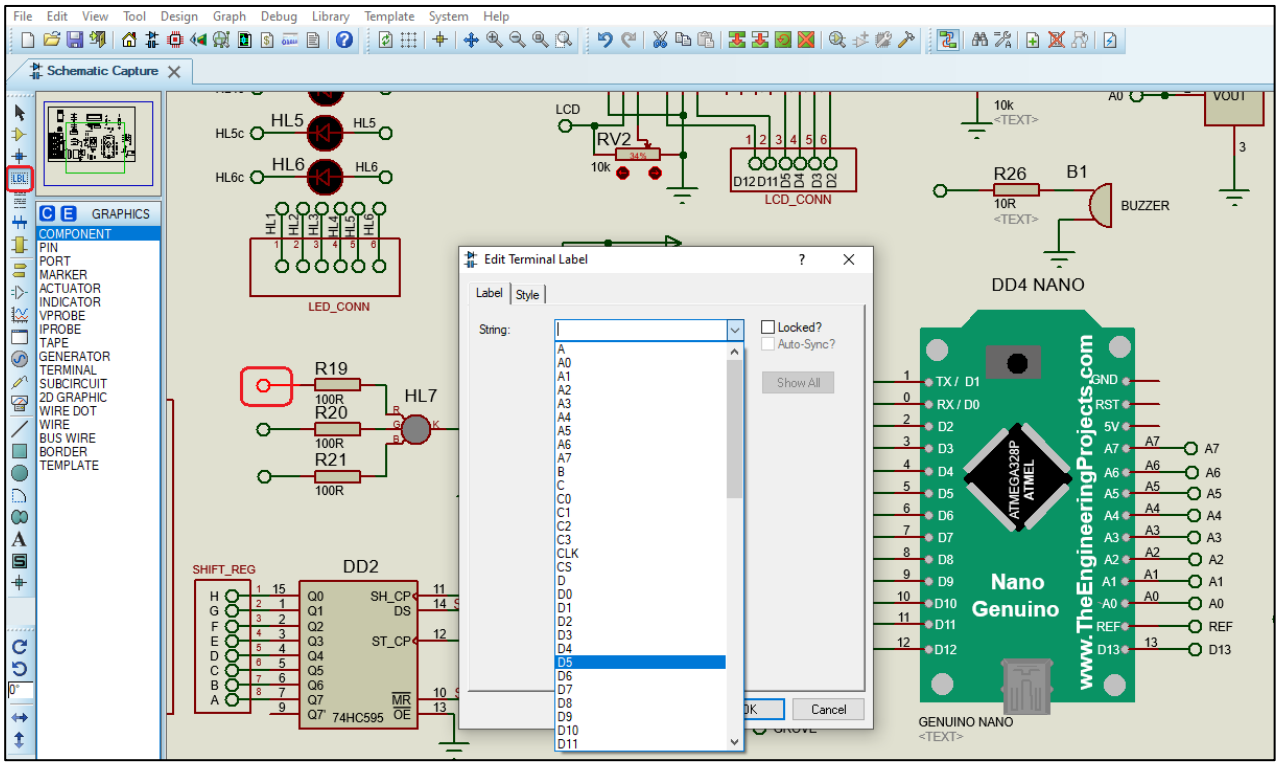


Рис.3.9. Підключення елементів до Arduino Nano в середовищі Proteus

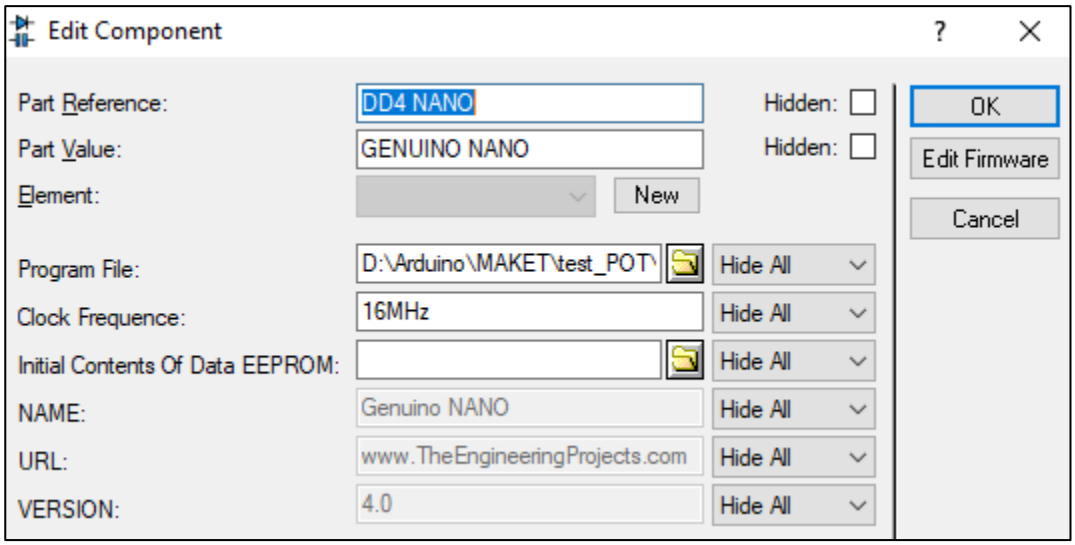


Рис.3.10 . Вікно властивостей мікроконтролера

Як альтернатива середовищу Proteus для моделювання роботи елементів схеми лабораторного макету можна скористуватись онлайн

програмним засобом Tinkercad Circuits [5, 6] (рис.3.11) або додатком UnoArduSim V2.6 [7, 8] (рис.3.12). Обидва додатка є безкоштовними симуляторами Arduino Uno, що надають можливість побачити хід виконання програми в реальному часі без наявності самої плати Arduino Uno.

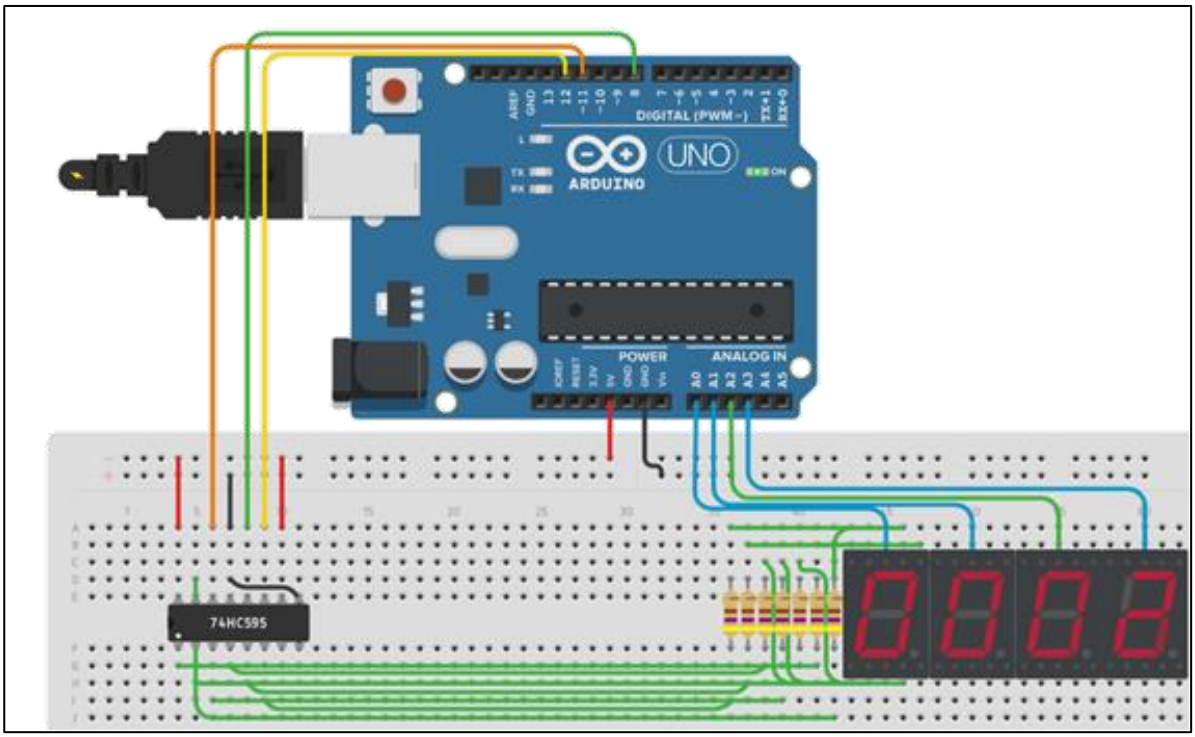


Рис.3.11. Моделювання роботи регістру зсуву 74HC595 та семисегментного індикатора у середовищі Tinkercad Circuits

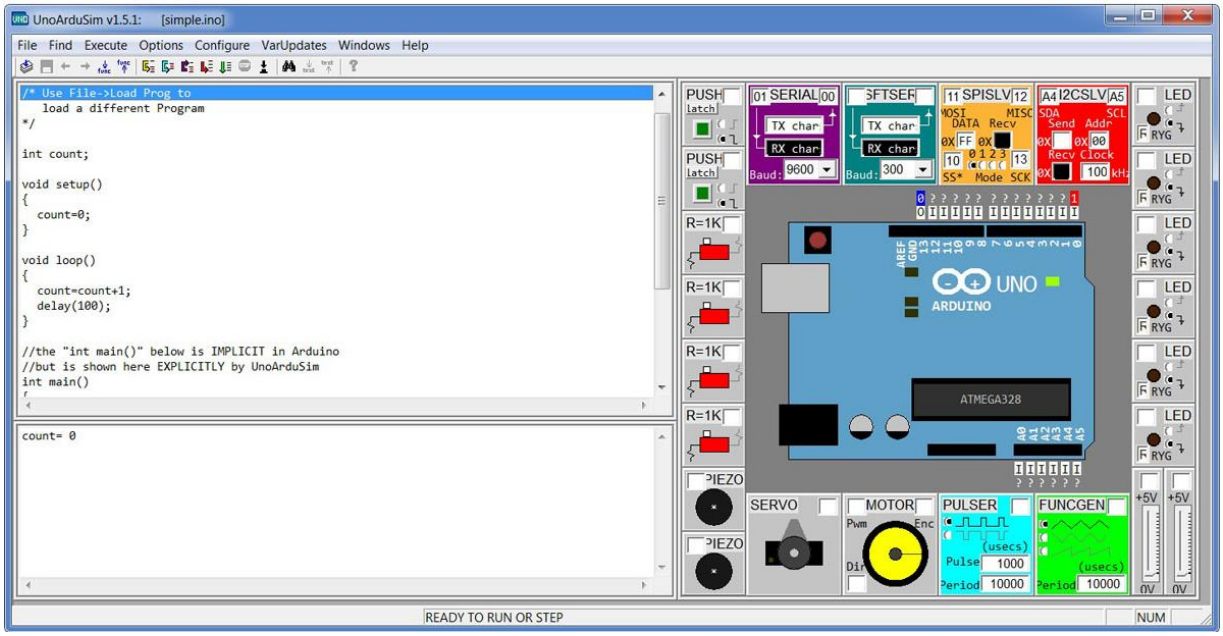


Рис.3.12. Вікно програми UnoArduSim

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 С 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.

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).

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 Т1, спадаючий
1	1	1	Зовнішній вхід Т1, наростаючий

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 . дисплеем 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.шістнадцятькова.

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.3.13).

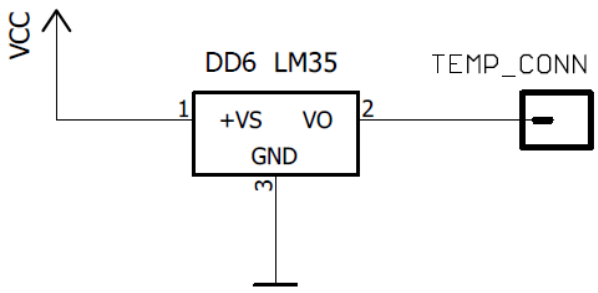


Рис.3.13. Схема підключення датчика LM35

Додаткові бібліотеки для Arduino не потрібні. **Приклад** програми робо ти 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 °С.

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.

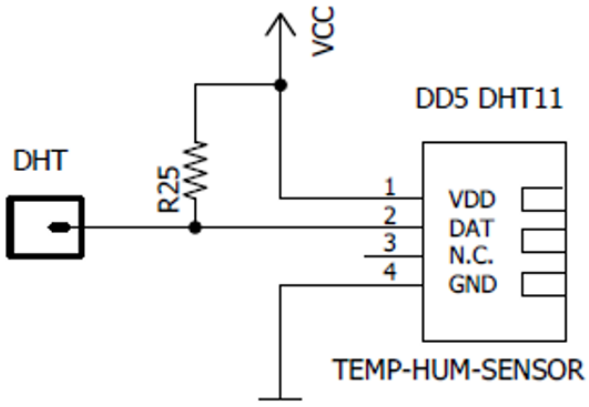


Рис.3.14 . Схема підключення датчика DHT11

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.3.14). 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.

Як показано на рис.3.15, 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.

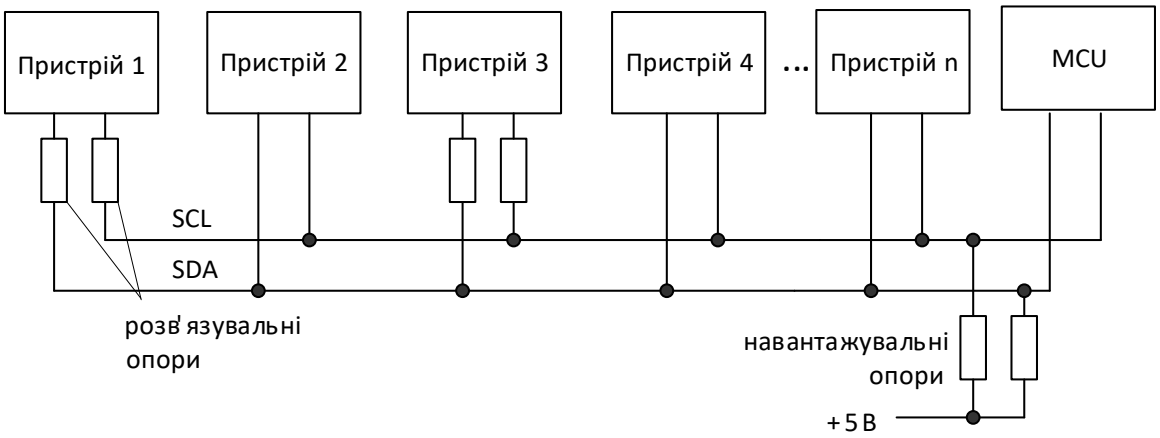


Рис.3.15 . Система протоколу TWI

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.3.16).

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.3.15).

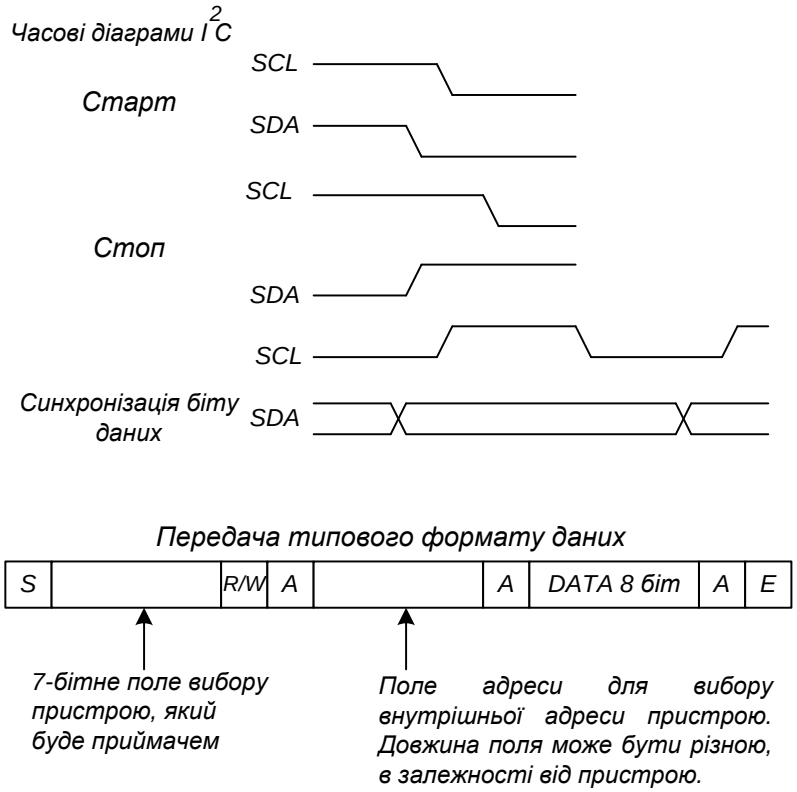


Рис.3.16. Часові діаграми роботи шини I^2C

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.3.17).

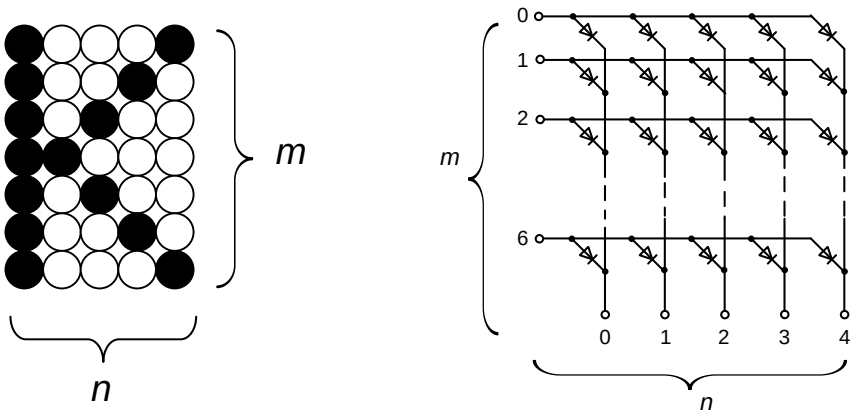


Рис.3.17 . Загальний вигляд та схема матричного індикатора

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.3.18.

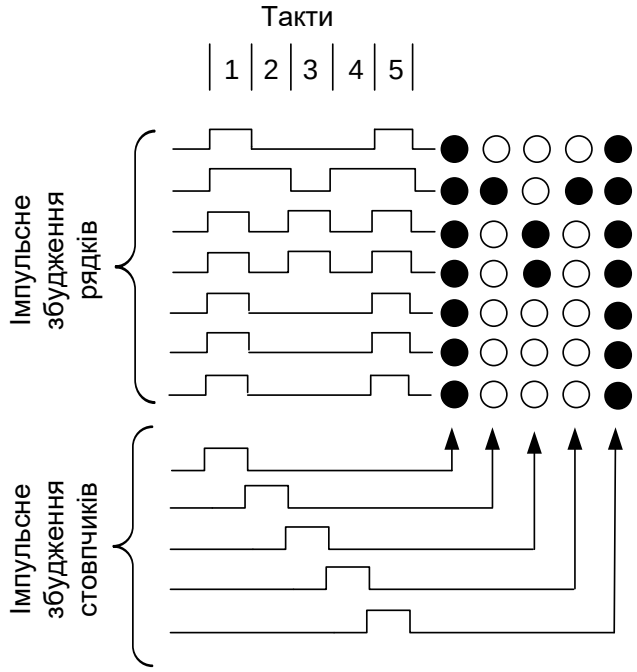


Рис.3.18 . Часова діаграма формування літери М

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 проініціалізирует 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 LedControl, *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.

6. Реалізувати програму, яка передбачає використання світлодіодної матриці. На світлодіодної матриці відобразити графік функції $2\sin x - x^2 + 2 = 0$ на відрізку [– 2; 3]. Для побудови графіку застосувати метод відокремлення коренів [19, с. 209-211].

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

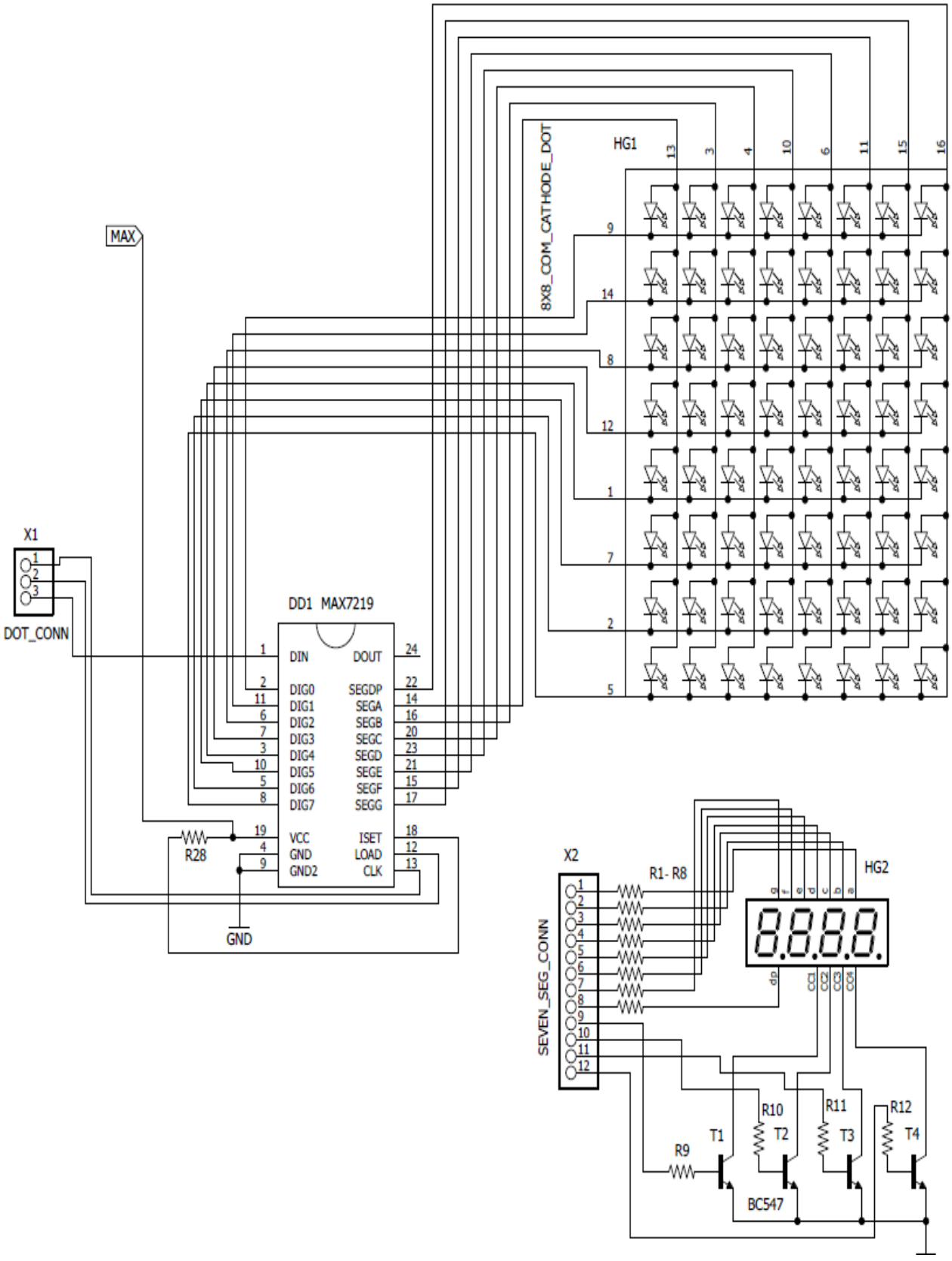
1. Цирульник С. М., Азаров О. Д., Крупельницький Л. В., Трояновська Т. І. Мікропроцесорна техніка: навчальний посібник. Вінниця: ВНТУ, 2017. 123 с.
2. Цирульник С. М. Лисенко Г. Л. Проектування мікропроцесорних систем. Вінниця: ВНТУ, 2012. 191 с.
3. Краткий учебный курс PROTEUS Руководство для начинающих. 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:

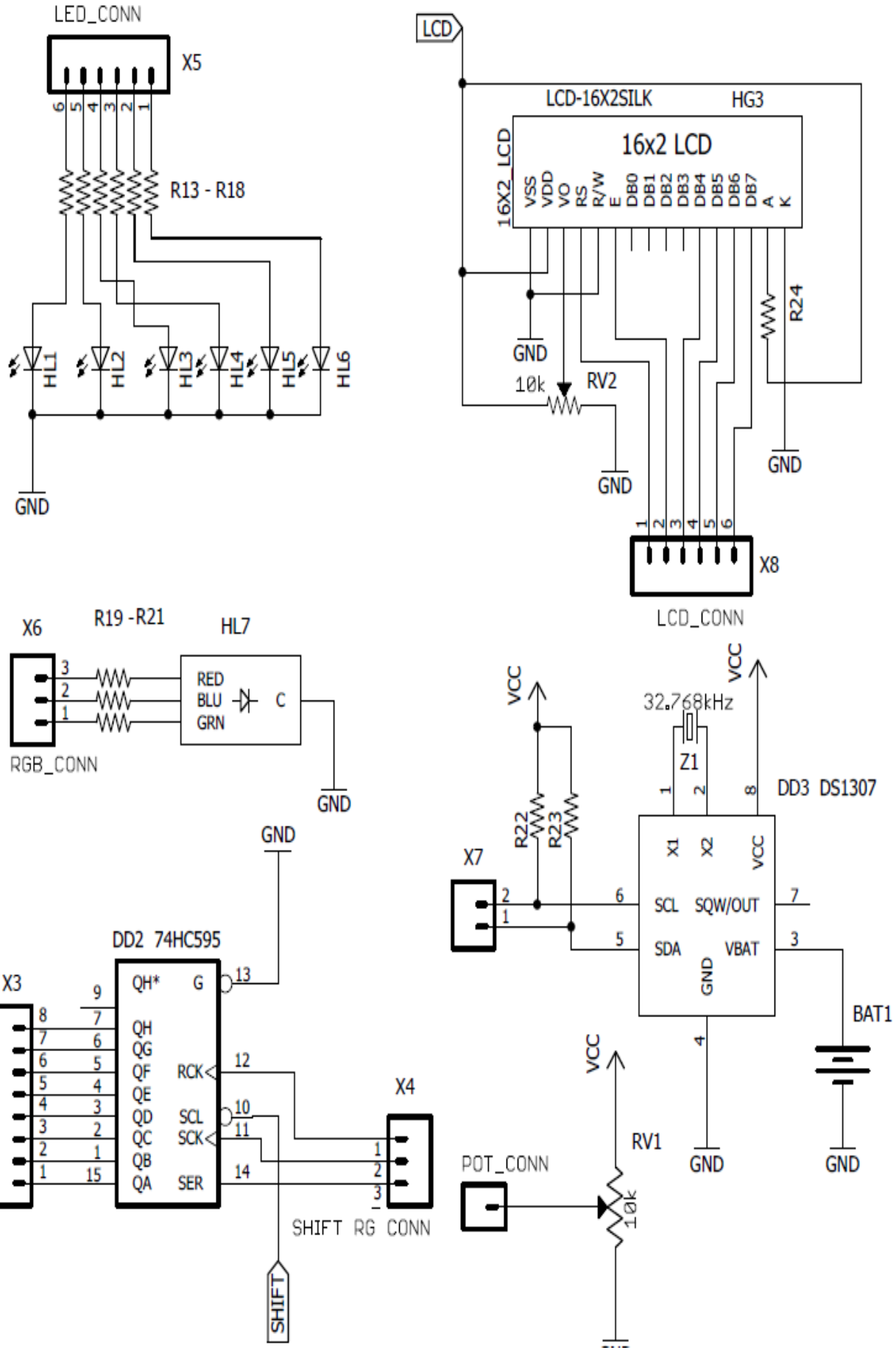
https://alexgyver.ru/lessons/arduino-reference.

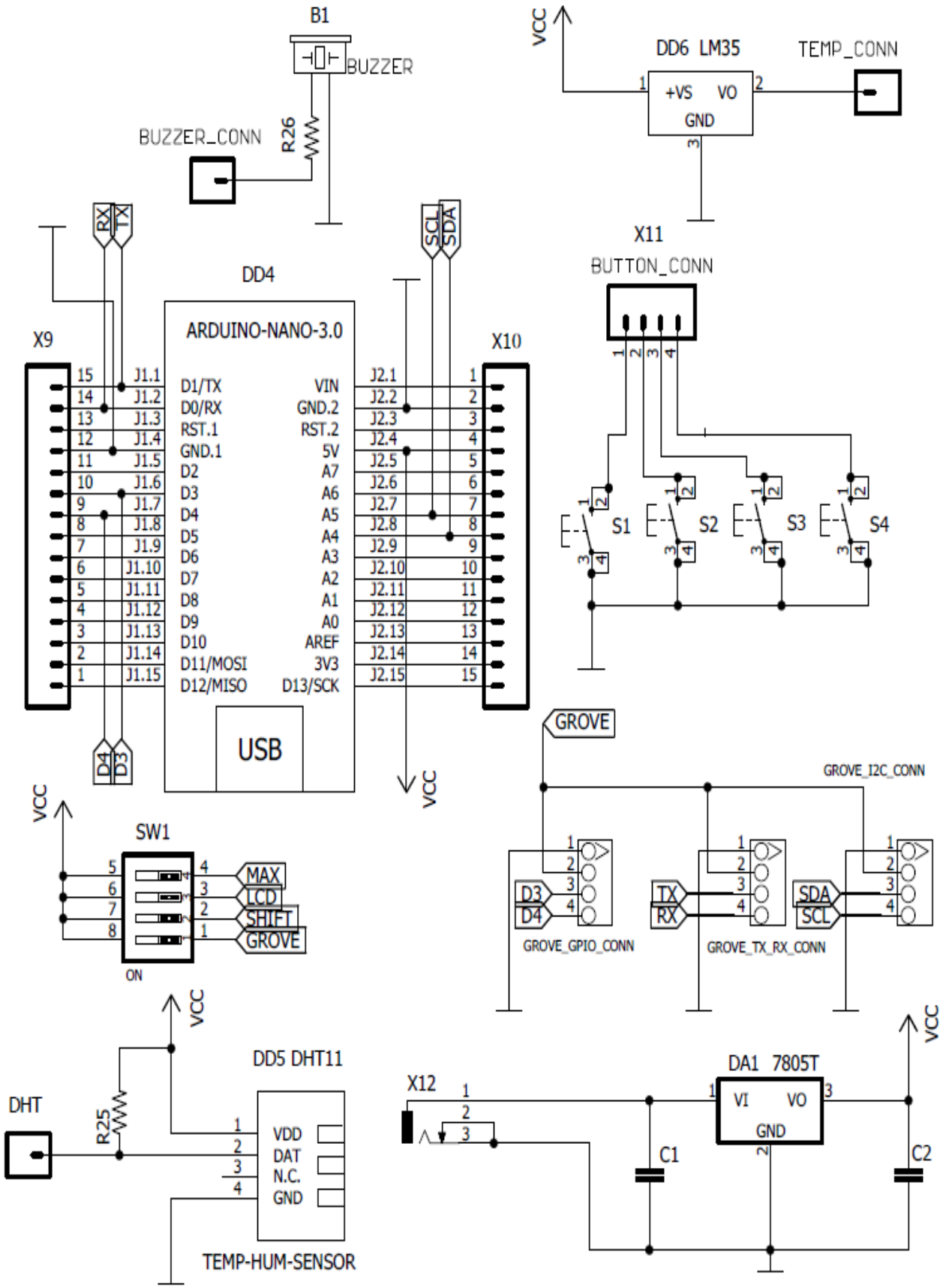
15. Программы для программирования плат Arduino и Digispark. URL: https://kolotushkin.com/indexsoft.php.
16. Библиотека LiquidCrystal. URL: http://developer.alexanderklimov.ru/arduino/liquidcrystal.php
17. LED-матрица 8×8. URL: http://developer.alexanderklimov.ru/arduino/ledmatrix.php
18. Визуальный редактор для создания анимационных эффектов LED Matrix Editor. URL: https://xantorohara.github.io/led-matrix-editor.
19. Волонтир Л.О, Зелінська О.В., Потапова Н.А., Чіков І.А. Чисельні методи: Навчальний посібник. Вінницький національний аграрний університет. Вінниця: ВНАУ, 2020. 322 с.

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
DD2 74HC595
X3
QH* G
QH
QG
QF RCK
QE
QD SCL
QC SCK
QB
QA SER
SHIFT_REG
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
ARDUINO-NANO-3.0
X9
X10
X11
BUTTON_CONN
S1
S2
S3
S4
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
GROVE
GROVE_I2C_CONN
GROVE_GPIO_CONN
GROVE_TX_RX_CONN
SW1
MAX
LCD
SHIFT
GROVE
ON
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 «С».

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 «°С».

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°C, 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*