

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

МІКРОПРОЦЕСОРНА ТЕХНІКА

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються за
спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані
технології», освітня програма «Технічні та програмні засоби
автоматизації»*

Київ
КПІ ім. Ігоря Сікорського
2021

Мікропроцесорна техніка: Комп'ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / КПІ ім. Ігоря Сікорського ; уклад. М. В. Коржик. Електронні текстові дані (1 файл 1 МБайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 47 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 8 від 24.06.2021 р.)
за поданням Вченої ради Інженерно-хімічного факультету (протокол № 5 від 24.05.2021 р.)*

Електронне мережне навчальне видання

МІКРОПРОЦЕСОРНА ТЕХНІКА

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

Укладач	Коржик Михайло Володимирович, канд. техн. наук, доцент
Відповідальний редактор	Жученко А. І., завідувач кафедри технічних та програмних засобів автоматизації, докт. техн. наук, професор
Рецензент	Степанюк А. Р., канд. техн. наук, доцент кафедри машин та апаратів хімічних та нафтопереробних виробництв інженерно-хімічного факультету КПІ ім. Ігоря Сікорського

Запропонований навчальний посібник містить матеріали для проведення робіт комп'ютерного практикуму та виконання розрахунково-графічної роботи з розділу «Мікроконтролери» кредитного модуля «Мікропроцесорна техніка».

Призначений для студентів спеціальності «Автоматизація та комп'ютерно-інтегровані технології» всіх форм навчання.

© КПІ ім. Ігоря Сікорського, 2021

Зміст

	Стор.
Вступ.....	4
Робота 1. Властивості мікроконтролера ATmega8	5
Робота 2. Імітаційне моделювання електронних схем	11
Робота 3. Пристрої відображення символічної інформації	14
Робота 4. Програмування мікропроцесорних контролерів	18
Робота 5. Формування растрових зображень	21
Розрахунково-графічна робота	27
Список рекомендованої літератури	29
Додаток А. Властивості 8-розрядних AVR мікроконтролерів	30
Додаток Б. Система команд мікроконтролера ATmega8.....	31
Додаток В. Службові регістри мікроконтролера ATmega8.....	36
Додаток Г. Паралельний порт AVR-контролера	38
Додаток Д. Формат файлу прошивки AVR-контролера	39
Додаток Е. Операції з бітами у мові С	41
Додаток Ж. Завдання для програмування мікроконтролера	44
Додаток К. Приклад виконання блок-схеми алгоритму керування ...	47

Вступ

Сучасні мікроконтролери є основою техніко-програмного забезпечення керуючих пристроїв промислової автоматики.

Цей практикум призначено для формування у студентів напрямку «Автоматизація та комп'ютерно-інтегровані технології» цілісного уявлення про сучасні мікропроцесорні контролери та отримання знань і вмінь, необхідних для створення пристроїв керування на базі мікроконтролерів та їх програмування у відповідності до поставленої задачі. Виконання робіт комп'ютерного практикуму здійснюється за допомогою популярної системи автоматизованого проектування Proteus VSM (Labcenter Electronics, Великобританія) на прикладі широко розповсюдженого мікроконтролера початкового рівня ATmega8 (Atmel, Microchip Technology, США).

В загальному сенсі мікроконтролер – це програмно керована інтегральна мікросхема, яка використовується для побудови різного роду керуючих пристроїв. Крім обчислювального ядра (процесора) мікроконтролер зазвичай містить тактовий генератор, пам'ять та деякий набір периферійних пристроїв (в тому числі спеціалізованих), розташованих на тому ж кристалі. Такий підхід дозволяє істотно знизити розміри, енергоспоживання та вартість пристроїв, побудованих на базі мікроконтролера (однокристального комп'ютера) та скоротити час на проектування та впровадження готових проектів.

Вибір мікроконтролера ATmega8 для виконання робіт пропонованого комп'ютерного практикуму зумовлений тим, що ця відносно проста та дешева мікросхема містить багатий асортимент універсальних периферійних пристроїв, має порівняно великий і різноманітний набір команд та забезпечує високу продуктивність при низькому енергоспоживанні. Завдяки цьому ATmega8 свого часу стала базовою для побудови відомої апаратно-програмної платформи Arduino (Італія),

початково призначеної саме для задач ознайомлення студентів із основами мікропроцесорної техніки [3, 16].

Крім того всі мікроконтролери Atmel добре забезпечені безкоштовною технічною документацією, засобами програмування (для різних операційних систем) та великою кількістю навчальних і методичних матеріалів, що сприяє їх широкій популярності.

Робота 1

ВЛАСТИВОСТІ МІКРОКОНТРОЛЕРА ATmega8

Мета роботи: Дослідити склад та функціональні можливості мікроконтролера ATmega8.

Теоретичні відомості

ATmega8 належить до 8-розрядний AVR контролер сімейства Mega і може працювати із тактовою частотою до 16 МГц. Напруга живлення контролера в залежності від модифікації варіюється від 2.7 до 5.5 В.

AVR архітектура характеризується поєднанням класичного підходу до побудови обчислювального ядра типу CISC з деякими особливостями RISC процесорів (велика кількість регістрів загального призначення, виконання більшості команд за 1 машинний цикл тощо), та використанням гарвардської моделі пам'яті (роздільний доступ до області даних та програмної області), що забезпечує високу продуктивності при порівняно низькому ступені інтеграції елементів мікросхеми. Властивості AVR контролерів сімейств Mega та Tiny наведено у дод. А.

Контролер, що розглядається, споряджено 8 КБ (див. цифру в назві моделі) вбудованої Flash-ROM пам'яті для зберігання програмного коду (10000 циклів перезапису), та 1 КБ SRAM – оперативної пам'яті.

Систему команд контролера ATmega8 наведено у дод. Б.

Периферія мікроконтролера містить наступні пристрої (розпіновка ATmega8 зображена на рис. 1):

- 23 лінії GPIO (загального призначення), організовані як три паралельні порти (виводи: 8 ліній **PBx**, 7 ліній **PCx** та 8 ліній **PDx**).

- 8-бітний таймер/лічильник **TC0** з функцією Overflow (імпульсів на рахунковому вході **T0**).

- 8-бітний таймер/лічильник **TC2** з функціями Overflow і Output Compare (вивід **OC2**) та підтримкою PWM (1 канал).

- 16-бітний таймер/лічильник **TC1** з функціями Overflow (імпульсів на T1), Output Compare (виводи **OC1A** та **OC1B**), Input Capture (за сигналом на виводі **ICP1**) та підтримкою PWM (2 канали).

- Вартовий таймер (Watchdog).

- 8-канальний АЦП (із роздільною здатністю 10 бітів, виводи **ADCx**).

- Компаратор аналогових сигналів (виводи **AIN0** та **AIN1**).

- Послідовний інтерфейс SPI (виводи **MISO**, **MOSI**, **SCK** та **SS**).

- Послідовний інтерфейс USART (виводи **XCK**, **TXD** та **RXD**).

- Послідовний інтерфейс TWI (виводи **SCL** та **SDA**).

- Додатковий енергонезалежний запам'ятовуючий пристрій типу EEPROM обсягом 512 Б (100000 циклів перезапису).

Вказані периферійні пристрої є джерелами внутрішніх переривань (до 16). Крім того контролер підтримує три зовнішні переривання (**Reset**, **INT0** та **INT1**).

Службові регістри периферійних пристроїв а також регістри спеціальних функцій SFR (Special Function Registers) зосереджені у ізольованій області так званих портів введення / виведення (**P**) обсягом 64 Б (див. дод. В), хоча лише деякі з цих регістрів є справжніми портами. В AVR контролерах до службових регістрів також можна звернутися як до звичайних комірок, що розташовані у загальному просторі пам'яті даних контролера (систему команд див. у дод. Б.).

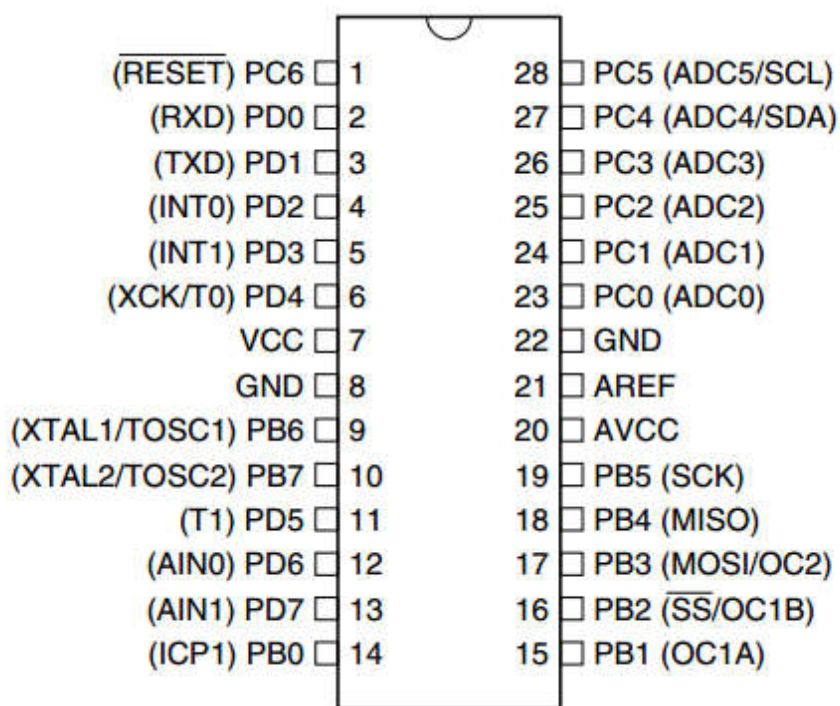


Рис. 1. Схема розташування виводів АТmega8 у PDIP виконанні

Будь-який процесор потребує джерела тактових імпульсів (Clock Pulses). Контролер, що розглядається, може працювати з наступними джерелами:

- Тактовий генератор на базі вбудованого RC-ланцюга 8 МГц (з переддільником на 1, 2, 4 та 8). Має невисоку точність та стабільність ($\pm 5\%$). За замовченням контролер налаштований на роботу з частотою 1 МГц від внутрішнього RC-генератора (див. таб. 1 та 2).
- Тактовий генератор на базі зовнішнього RC-ланцюга (виводи **XTAL1** та **GND**), використовується для роботи на нестандартних частотах (до 12 МГц) без вимог до високої точності.
- Тактовий генератор на базі зовнішнього резонатора (виводи **XTAL1** та **XTAL2**) – кварцевого (максимальна похибка $\pm 0.01\%$) або керамічного ($\pm 0.5\%$). Наприклад, для організації годинника реального часу, як правило, використовують низькочастотний кварцевий резонатор 32.768 кГц (або 2^{15} Гц).
- Зовнішній тактовий генератор (вивід **XTAL1**), зазвичай для синхронної

роботи декількох окремих цифрових пристроїв.

Вибір джерела та частоти тактових імпульсів належить до процедур початкового низькорівневого програмування контролера і здійснюється за допомогою fuse-регістрів, розташованих у Flash-ROM пам'яті. ATmega8 має два 8-розрядні fuse-регістри. Доступ до регістрів здійснюється командами **LPM** та **SPM** як до комірок програмної пам'яті за адресами 0x0000 та 0x0003 при дотриманні відповідних алгоритмів захисту. Призначення бітів fuse-регістрів наведено у табл. 1.

Таблиця 1

Біт	Призначення	Початкове значення	
STDISBL	Reset на виводі PC6	1	Reset присутній
WDTON	вимкнення Watchdog	1	можна вмикати і вимикати
SPIEN	програмування через SPI	0	дозволено
CKOPT	при наявності зовнішнього резонатора	1	максимально 16 МГц (при 0 максимально 8 МГц)
EESAVE	стирання EEPROM при програмуванні	1	дозволено
BOOTSZ1	розмір (та адреса) сектора завантажувача	0	00 – 1024 слова (01 – 512, 10 – 256, 11 – 128 слів)
BOOTSZ0		0	
BOOTRST	після Reset перейти на адресу завантажувача	1	ні, перейти на нульову адресу
BODLEVEL	рівень спрацьовування детектора 2.7 / 4 В	1	2.7 В (0 – 4 В)
BODEN	детектор неприпустимого рівня напруги живлення	1	вимкнений
SUT1	вибір затримки запуску після появи живлення	1	0, 4, 65 мс (в залежності від комбінації CKSELx)
SUT0		0	
CKSEL3	режим роботи тактового генератора	0	Внутрішній RC (1, 2, 4, 8 МГц), зовнішній RC (до 12 МГц), зовнішній резонатор, зовнішня синхронізація.
CKSEL2		0	
CKSEL1		0	
CKSEL0		1	

Налаштування джерела тактових імпульсів (встановлення бітів **SUT_x** та **CKSEL_x**) виконується відповідно до DataSheet контролера [16], або за допомогою fuse-калькулятора (наприклад, https://payalo.at.ua/c_fuse/calc).

Деякі режими тактування контролера ATmega8 наведено у табл. 2.

Слід відзначити, що в AVR контролерах встановленим (запрограмованим) значенням fuse-біта вважається рівень **0**. Рівень **1** відповідає скинутому (незапрограмованому) стану. Ця особливість є причиною багатьох помилок при програмуванні за допомогою різних популярних застосунків для прошивки, що може призвести до виведення контролера з ладу.

Таблиця 2

SUT₁	SUT₀	CKSEL₃	CKSEL₂	CKSEL₁	CKSEL₀	Режим тактування контролера
0	0	0	0	0	0	Зовн. синхронізація, затримка 6 т + 0 мс
0	1	0	0	0	0	Зовн. синхронізація, затримка 6 т + 4 мс
1	0	0	0	0	0	Зовн. синхронізація, затримка 6 т + 65 мс
0	0	0	0	0	1	Внутр. RC, 1 МГц, затримка 6 т + 0 мс
0	1	0	0	0	1	Внутр. RC, 1 МГц, затримка 6 т + 4 мс
1	0	0	0	0	1	Внутр. RC, 1 МГц, затримка 6 т + 65 мс
0	0	0	0	1	0	Внутр. RC, 2 МГц, затримка 6 т + 0 мс
0	1	0	0	1	0	Внутр. RC, 2 МГц, затримка 6 т + 4 мс
1	0	0	0	1	0	Внутр. RC, 2 МГц, затримка 6 т + 65 мс
0	0	0	0	1	1	Внутр. RC, 4 МГц, затримка 6 т + 0 мс
0	1	0	0	1	1	Внутр. RC, 4 МГц, затримка 6 т + 4 мс
1	0	0	0	1	1	Внутр. RC, 4 МГц, затримка 6 т + 65 мс
0	0	0	1	0	0	Внутр. RC, 8 МГц, затримка 6 т + 0 мс
0	1	0	1	0	0	Внутр. RC, 8 МГц, затримка 6 т + 4 мс
1	0	0	1	0	0	Внутр. RC, 8 МГц, затримка 6 т + 65 мс

Справа в тому, що деякі застосунки (утиліти) для прошивки мікроконтролерів використовують інверсну логіку при зверненні до fuse-

регістрів. Інколи в такій програмі є можливість перемикання типу логіки, що лише погіршує ситуацію.

При під'єднанні до мікроконтролера за допомогою послідовного програматора завжди слід звертати увагу на біт **SPIEN** (який повинен мати дозвільне значення, оскільки ви вже успішно прочитали вміст fuse-регістрів) і формувати інші біти у відповідності з цією логікою.

Завдання

1. Приєднати контролер ATmega8 за допомогою послідовного програматора (наприклад, USB ISP-ASP) до комп'ютера.
2. Прочитати вміст fuse-регістрів контролера за допомогою декількох безкоштовних утиліт для прошивки (наприклад, AVRdude, eXtreme Burner, Khazama AVR Programmer, PonyProg тощо) та порівняти результати.
3. Визначити поточне налаштування контролера на джерело та частоту тактових імпульсів.
4. Встановити на контролері режим роботи від внутрішнього RC-генератора на частоті 8 МГц з максимальною затримкою запуску (Start-up Time, див. табл. 2).

Контрольні запитання

1. Які периферійні пристрої містить ATmega8?
2. Як приєднати контролер до джерела живлення?
3. Що таке fuse-біти контролера?
4. Як приєднати контролер ATmega8 до послідовного програматора?
5. Як прочитати вміст області налаштувань контролера?

Робота 2

ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ СХЕМ

Мета роботи: Ознайомитись з можливостями та принципами роботи системи схемотехнічного моделювання Proteus.

Теоретичні відомості

Сьогодні існує велика кількість програмних продуктів (в тому числі безкоштовних та умовно-безкоштовних) для емуляції електронних схем та проектування друкованих плат електронних пристроїв (Autodesk Circuits, NI Multisim, DesignSparkPCB, KiCAD, MicroSim PSpice, OrCAD тощо).

Для моделювання схем на основі мікроконтролерів великою популярністю користується система автоматизованого проектування Proteus Design (див. <https://www.labcenter.com/downloads/>). Система використовує мову програмування SPICE (університет Берклі, США) для опису та аналізу поведінки компонентів електронної схеми (у специфікації XML).

Такий підхід використовується в багатьох подібних програмах, наприклад, в продуктах компанії Cadence Design Systems (Сан-Хосе, США), але на відміну від останніх, бібліотеки Proteus, навіть у безкоштовній версії, містять велику кількість готових примітивів поширених електронних компонентів (в тому числі мікроконтролерів), обладнаних імітаційними моделями, створеними безпосередньо виробниками цих пристроїв.

Більшу частину вікна програми емуляції (ISIS) займає масштабована робоча область **Editing Window** на якій розташовуються графічні примітиви компонентів та здійснюється їх з'єднання. Аркуш проєкту на робочій області позначено синім прямокутником (розміри задаються в меню **System / Set Sheet Size**), хоча створювати схеми можна і поза межами аркуша [2].

Орієнтуватися в робочій області допомагає вікно попереднього перегляду **Overview Window**, під яким розташовані засоби вибору об'єктів

Object Selector (компонентів, пристроїв, інструментів, джерел живлення тощо). Різні категорії об'єктів вибираються з відповідних меню. При необхідності потрібні об'єкти додаються в ці меню із зовнішніх бібліотек за допомогою кнопки **Pick** (якою обладнано кожне меню).

Основні компоненти для формування схеми накопичуються у меню **Devices** та встановлюються на робочу область. Для цього треба вибрати компонент у меню та скопіювати його у робочу область натисканням лівої клавіши миші (ЛК). Обраний компонент закріплюється у потрібному місці робочої області повторним ЛК (підтвердження дії). Переміщення вже встановленого компонента здійснюється також ЛК після наведення курсора на примітив. Подвійне ЛК викликає вікно редактора властивостей компонента **Edit Component**. Натискання правої клавіши миші (ПК) на примітив зазвичай викликає його контекстне меню. Повторне ПК вибраний компонент видаляє (скасування дії). При необхідності у режим вибору компонентів можна перейти за допомогою кнопки **Selection**.

З'єднання елементів здійснюється вибором початку лінії зв'язку за допомогою ЛК, прокладанням цієї лінії курсором миші та фіксацією завершення лінії повторним ЛК. Якщо увімкнена функція **Wire Autorouter** (кнопкою на панелі інструментів), прокладання лінії зв'язку відбувається у відповідності з правилами автотрасування.

Система підтримує віртуальне під'єднання шин живлення мікросхем (VCC або VDD та GND), але лише для задач імітаційного моделювання. Для проектування реальних друкованих плат відповідні елементи треба встановлювати з меню **Terminals** та під'єднувати самотійно.

Керування імітацією роботи створеної електронної схеми здійснюється кнопками у нижній частині вікна. Для зміни параметрів компонентів імітацію схеми треба зупиняти.

Слід відзначити, що не всі бібліотечні компоненти мають імітаційну модель (такі об'єкти для емуляції роботи схем не використовуються).

Компоненти з деяких бібліотеки (наприклад, *ASimMDLs*, *Display*, *Device*, *Active* тощо) є віртуальними, вони не мають креслень корпусів але частина з них обладнана елементами анімації для наочної демонстрації їх роботи при емуляції.

Завдання

1. У вікні графічного редактора Proteus ISIS налаштувати робочу область з розміром аркуша А4 у альбомній орієнтації.

2. Відпрацювати прийоми пошуку компонентів, формування та імітації електронної схеми, на прикладі схеми роботи сигнального світлодіода (див. рис. 2).

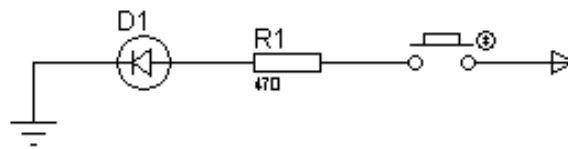


Рис. 2. Схема увімкнення світлодіода

Для роботи схеми можуть знадобитися компоненти: Button або Switch, Resistor, LED та термінальні елементи Power і Ground.

Контрольні запитання

1. Які програми імітаційного моделювання електронних схем вам відомі?

2. З яких модулів складається система автоматизованого проектування Proteus?

3. З яких об'єктів може формуватися імітаційна схема Proteus і де вони розташовані?

4. Яким чином можна встановити об'єкт у імітаційну схему Proteus?

5. Яким чином можна змінити властивості компонента, встановленого у імітаційну схему Proteus?

Робота 3

ПРИСТРОЇ ВІДОБРАЖЕННЯ СИМВОЛЬНОЇ ІНФОРМАЦІЇ

Мета роботи: Ознайомитись з роботою світлодіодних пристроїв візуалізації цифрової інформації.

Теоретичні відомості

Традиційно, найпростішим та найпоширенішим пристроєм відображення алфавітних символів у цифрових електронних пристроях є семисегментний індикатор (7-Seg), який насправді має вісім світлодіодних сегментів (наприклад, однорозрядний 7-Seg Ningbo Flying Electronics із спільним анодом СА або спільним катодом СС, див. рис. 3) і керується однобайтним цифровим кодом (TTL рівнів).

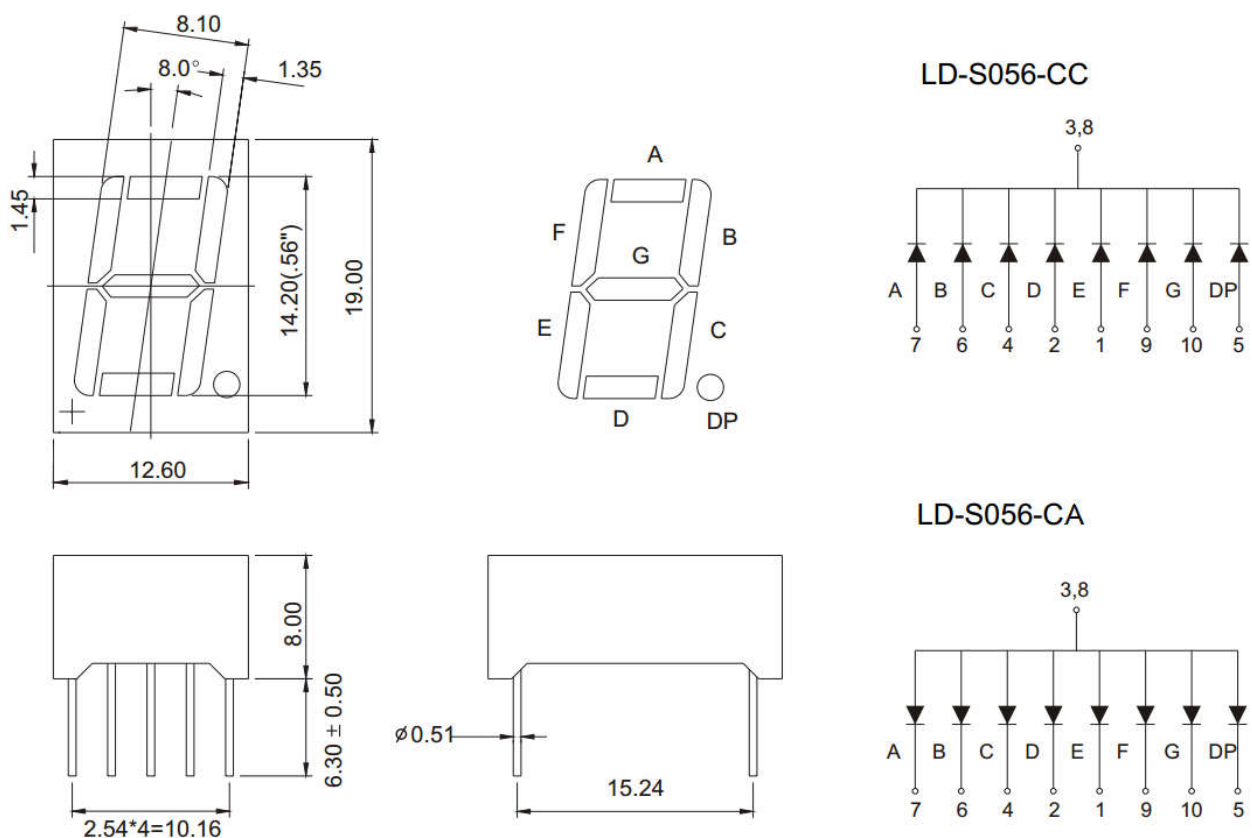


Рис. 3. Специфікація 7-Seg індикатора

Кожен сегмент, як і звичайний світлодіод, слід під'єднувати через обмежувальний резистор для забезпечення номінальної величини струму (зазвичай 20 мА для сигнальних світлодіодів, див. рис. 2).

Якщо логічний вхід у цифровому пристрої формується на відкритому виводі ключового елемента (наприклад, транзистора з відкритим колектором), або на високоімпедансному виводі (z-стан, тристабільний буфер), рівень цього сигналу може виявитись невизначеним і потребуватиме уточнення шляхом введення в схему підтягуючих резисторів (pull-up або pull-down, див. рис. 4). Така ситуація справедлива для мікросхем на базі TTL логіки, але часто зустрічається і у семисегментних індикаторах. При цьому час зростання або спаду сигналу до відповідного логічного рівня пропорційний величині опору підтягуючого резистора (зазвичай 10 кОм).

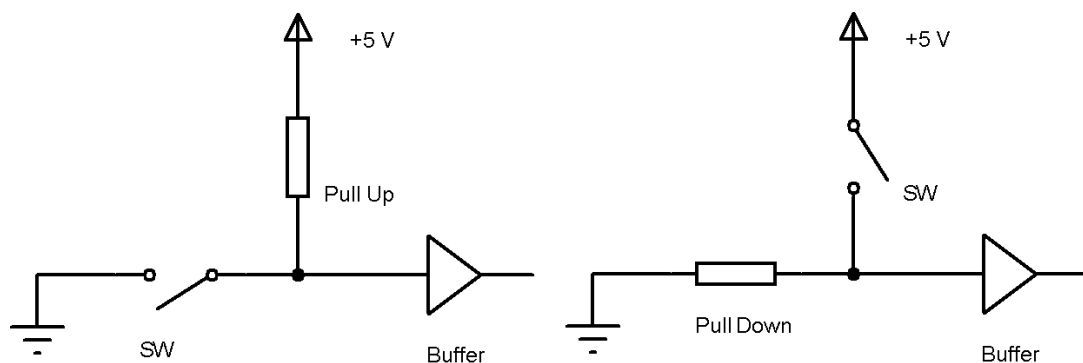


Рис. 4. Підтягуючий резистор

Для більш якісного відображення символічної інформації інколи використовують 14-и та 16-сегментні індикатори або світлодіодні матриці різної розмірності (наприклад, на рис. 5 зображено схему матриці Rayconn Electronics розмірністю 8x8).

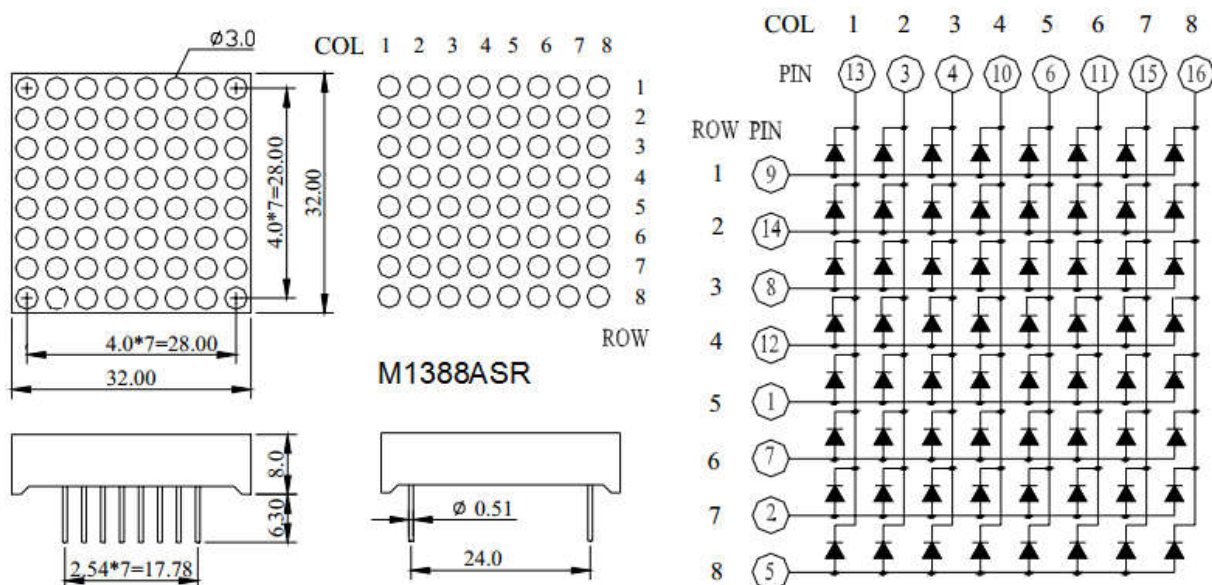


Рис. 5. Специфікація світлодіодної матриці 8x8

Світлодіодна матриця для увімкнення окремого сегмента потребує високого рівня напруги на відповідному аноді та низького рівня на відповідному катоді. Інші комбінації сигналів є недійсними. Під'єднувати піни стовпців або рядків матриці слід через обмежувальні резистори.

Завдання

За допомогою Proteus ISIS створити імітаційні моделі і дослідити роботу схем світлодіодної індикації символічної інформації:

1. Зібрати схему з чотирьох семисегментних індикаторів та за допомогою комутаційних елементів вивести на них шифр вашої навчальної групи (див. рис. 6).

2. Зібрати схему із світлодіодною матрицею 8x8, дослідити роботу матриці та вивести на неї за допомогою комутаційних елементів зображення, запропоноване викладачем (див. рис. 7).

Контрольні запитання

1. Які світлодіодні пристрої відображення символічної інформації вам відомі?
2. Які типи семисегментних індикаторів вам відомі?

3. Для чого використовуються підтягуючі резистори?
4. Охарактеризуйте принципи роботи світлодіодної матриці.
5. Які обмеження у виведенні інформації на світлодіодну матрицю існують при роботі у "статичному" режимі?

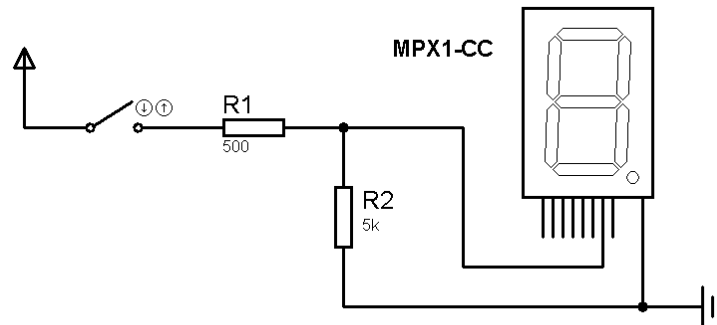


Рис. 6. Схема увімкнення 7-Seg індикатора

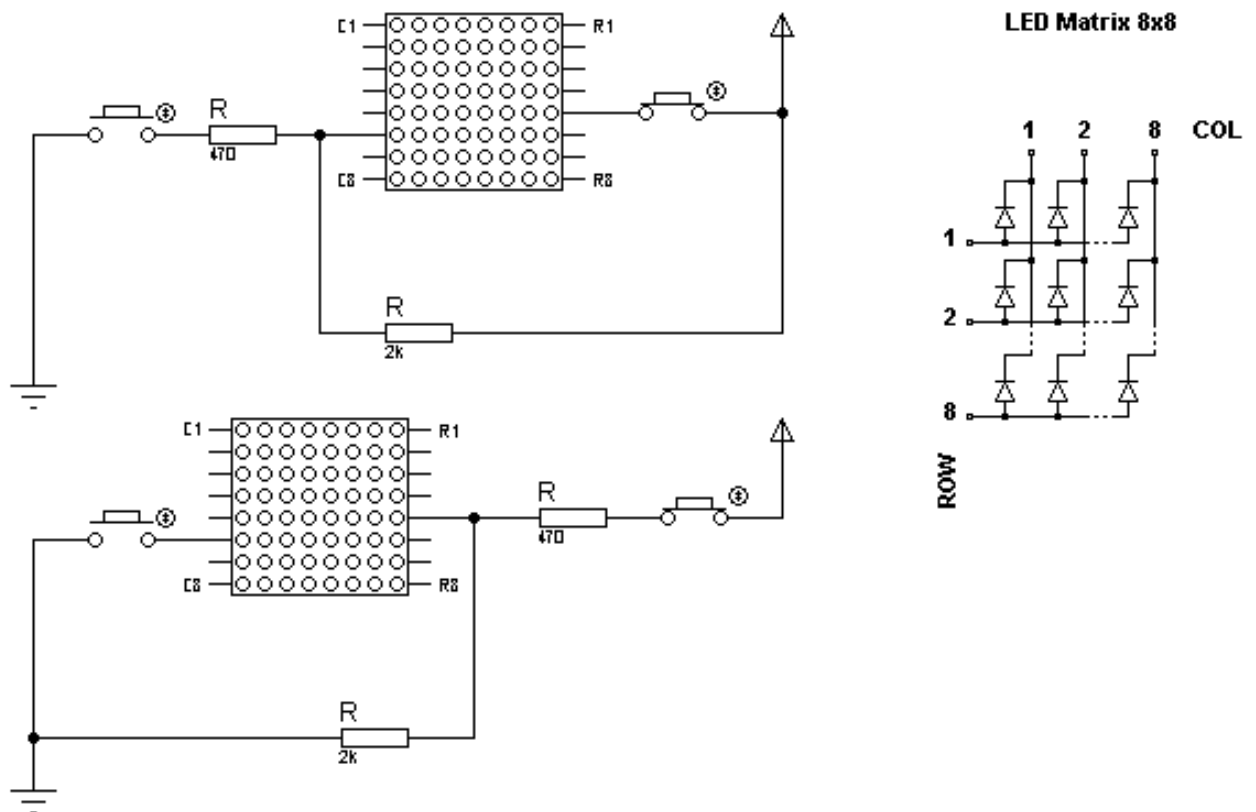


Рис. 7. Схема увімкнення світлодіодної матриці

Робота 4

ПРОГРАМУВАННЯ МІКРОПРЦЕСОРНИХ КОНТРОЛЕРІВ

Мета роботи: Ознайомитись з принципами програмування AVR контролерів на прикладі мікроконтролера ATmega8.

Теоретичні відомості

Існує велика кількість безкоштовних застосунків для програмування AVR-контролерів (наприклад, AVR Dude, WinAVR, CodeVisionAVR, MPLab тощо). У цьому практикуму ми будемо користуватися інтегрованим середовищем розробки Atmel Studio (яка до версії 6.0 мала назву AVR Studio, див. <https://www.microchip.com/en-us/development-tools-tools-and-software>). Цей застосунок базується на Microsoft Visual Studio і призначено для програмування мікроконтролерів Atmel архітектури AVR, AVR32 та ARM мовою C/C++ або безпосередньо на асемблері [8]. Atmel Studio містить редактор вихідного коду і засоби імітації та внутрішньосхемного відлагодження.

При створенні нового проєкту (**File/New/Project**) важливо дотримуватись наступної послідовності дій:

1. У вікні **New Project** обрати мову програмування: C (**GCC C Executable Project**) або асемблер (**AVR Assembler Project**).
2. Ввести ім'я проєкту та місце його розташування на диску (не рекомендується використовувати символи кирилиці).
3. У вікні **Board Selection** обрати модель контролера (в нашому випадку ATmega8 або ATmega8A). Тут одразу можна завантажити потрібний Datasheet.
4. Кнопкою **IO View** (на панелі інструментів) відкрити вікно ресурсів обраного контролера.

Наступні налаштування середовища Atmel Studio можуть істотно спростити роботу над проєктом. Рекомендується:

- Відключити **Show page on startup** на початковій сторінці.
- Закрити всі вікна праворуч крім **Solution Explorer**.
- Відімкнути у розділі **Underlines** вікна **Visual Assist Options** (відповідна кнопка ліворуч на панелі інструментів) функцію **Underline spelling** ..
- Через меню **Tools/Options..** у розділі
 - **Environment/General** відключити **Automatically adjust visual** ..
 - та **Enable rich** ..
 - **Environment/Fonts&Colors** встановити 12 або 14 пт.
 - **Text Editor** можна відімкнути **Track changer**.
 - **Text Editor/GCC** увімкнути **Line numbers**.
 - **Toolchain** виставити як **Atmel AVR 8-bit (C language)**.
- У вікні **Output** (знизу, кнопка додаткових опцій) можна увімкнути **Auto Hide**.

Сформуємо у середовищі ISIS (Proteus, див. роботу 2) імітаційну схему підключення світлодіодів на основі контролера ATmega8 (див. роботу 1), як це вказано на рис. 8.

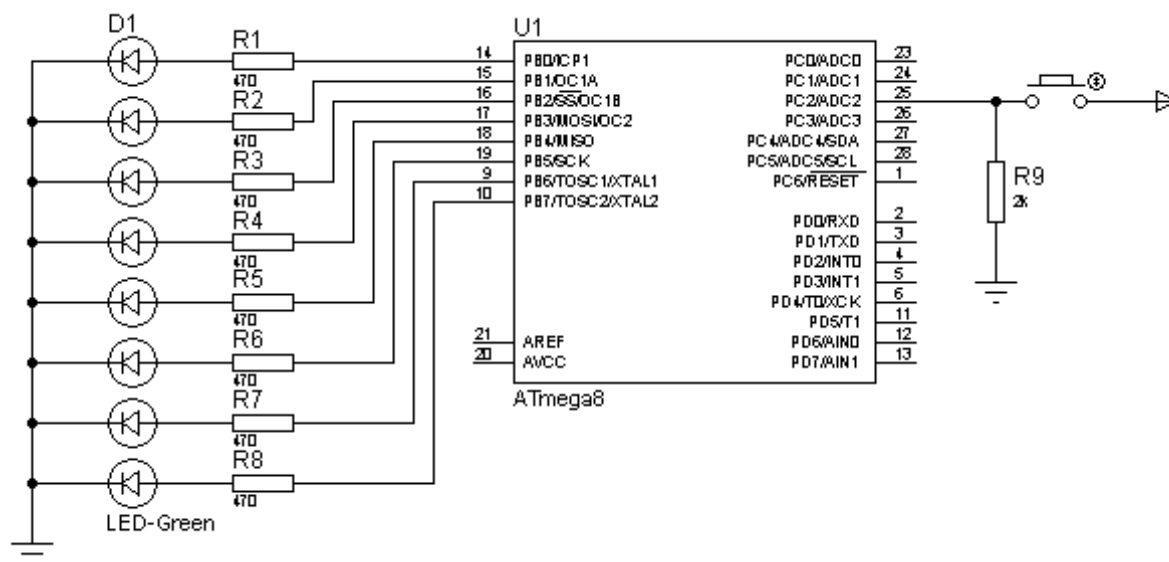


Рис. 8. Схема підключення світлодіодів до контролера ATmega8

Вісім світлодіодів тут під'єднано до всіх ліній порту В. Службові регістри для операцій з портами загального призначення (GPIO) наведено у

дод. В та Г. Створимо елементарну С-програму, що вмикає одночасно всі під'єднані світлодіоди.

// Приклад 1

```
#include <avr/io.h>

int main(void)
{
    DDRB = 0xFF;      //Port B - out
    PORTB = 0xFF;
    while(1) {
    }
```

У прикладі 1 увесь порт В налаштовано на режим Output та виведено високі рівні сигналів на відповідні лінії, що повинно призвести до вмикання під'єднаних світлодіодів. Далі у програмі розташовано нескінченний цикл **while** (формується автоматично при створенні проєкту в Atmel Studio), який є основою будь якої програми мікроконтролера.

Заголовний файл **io.h** присутній у всіх проєктах Atmel Studio, але його вміст залежить від обраної моделі контролера. У файлі реалізовано макровизначення, які дозволяють працювати з ресурсами контролера у термінах його технічної документації (Datasheet).

Після успішної компіляції у місці розташування проєкту (папка Debug) утворюються файли прошивок для мікроконтролера у форматах *.elf та *.hex. Останній формат (див. дод. Д) підтримують усі популярні утиліти для прошивки (див. роботу 1), хоча імітаційна модель контролера у Proteus може працювати і з іншими стандартами.

Для того, щоб завантажити програму у віртуальний мікроконтролер імітаційної схеми, треба відкрити редактор властивостей відповідного примітива (**Edit Component**, див. роботу 2) та вказати ім'я прошивки у розділі **Program File**. Інші розділи вікна редактора відповідають різним налаштуванням, доступним у фізичному мікроконтролері за допомогою

fuse-регістрів. Наприклад, для встановлення потрібної тактової частоти контролера слід використовувати розділ **CKSEL Fuses**.

Завдання

1. Для наведеної імітаційної схеми (рис. 8) організувати почергове циклічне увімкнення (обігання) під'єднаних світлодіодів.
2. Доповнити розроблену програму функцією перемикання напрямку обігання світлодіодів за допомогою під'єданого до однієї з ліній порту C пристрою комутації.

Інформацію щодо роботи з окремими бітами регістрів засобами мови C наведено у дод. Е.

Контрольні запитання

1. Поясніть призначення службових регістрів паралельних портів мікроконтролера ATmega8.
2. Як організувати встановлення окремих бітів 8-розрядного двійкового числа у мові C?
3. Як проаналізувати вміст окремих бітів 8-розрядного двійкового числа у мові C?
5. Як організувати введення сигналу, що відповідає логічному нулю, через лінію порту C (див. рис. 8)?

Робота 5

ФОРМУВАННЯ РАСТРОВИХ ЗОБРАЖЕНЬ

Мета роботи: Ознайомитись з принципами керування піксельними масивами за допомогою мікроконтролера.

Теоретичні відомості

Растрове зображення є впорядкованим масивом керованих пікселів (точок або елементарних фрагментів), розташованих на якомусь пристрої відображення. Важливими характеристиками зображення є його розмір

(кількість пікселів вздовж ширини та висоти графічної поверхні) та роздільна здатність (кількість пікселів на одиницю графічної поверхні).

Розглянемо формування растрових зображень на прикладі керування світлодіодною матрицею з розміром 8x8 (див. роботу 3). Зберемо у середовищі ISIS імітаційну схему, як це вказано на рис. 9.

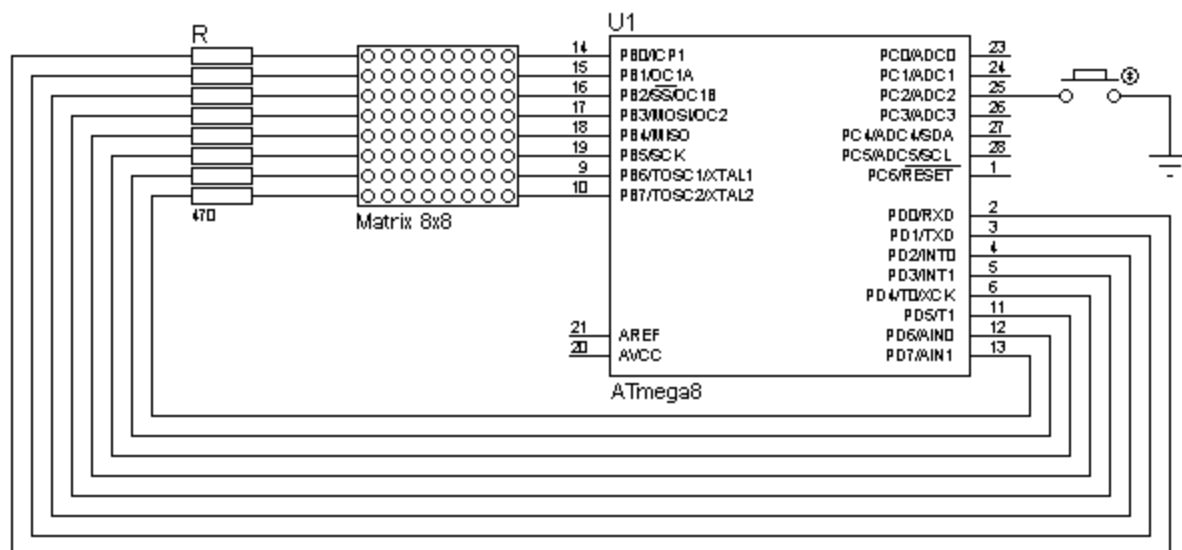


Рис. 9. Схема підключення світлодіодної матриці до контролера ATmega8

Виводи анодів та катодів у примітиві світлодіодної матриці треба визначати експериментально. У даному випадку до 8 ліній порту загального призначення **B** мікроконтролера ATmega8 під'єднано аноди матриці, що формують рядки (Raws) графічної поверхні, а до 8 ліній порту **D** – катоди, що формують стовпці (Columns). Порт **C** контролера містить лише 7 ліній (див. роботу 1) і може використовуватись для введення сигналів керування системою. У даному випадку до 3-ї лінії порту **C** під'єднано комутуючий пристрій, який забезпечує сигнал логічного 0 на виводі **PC2** контролера.

З роботи 3 відомо, як легко увімкнути окремий піксель матриці. Так само легко увімкнути декілька пікселів у межах одного рядка або одного стовпчика (статичне керування). Цей метод керування пікселями у різних рядках або стовпцях матриці одночасно можливий лише при виведенні у вказані рядки або стовпці однакової кодової комбінації (виведення

регулярного зображення), що істотно звужує потенційні можливості індикатора у 64 пікселі.

Модифікуємо програму з прикладу 1 (див. роботу 4) для формування на індикаторі регулярного зображення у вигляді заповненого квадрата, представленого на рис. 10а.

// Приклад 2

```
#include <avr/io.h>

int main(void)
{
    DDRB = 0xFF;          // Port B - Out
    DDRD = 0xFF;          // Port D - Out
    PORTB = 0b01111110;   // Row
    PORTD = 0b10000001;   // Col

    while(1) {
    }
```

Якщо треба вивести на індикатор лише контур квадрата, статичним методом зробити це *неможливо*, хоча можна вивести послідовно окремі ортогональні лінії (геометричні примітиви з яких складається зображення), і якщо це робити достатньо швидко, зоровий апарат людини може сприйняти серію як цільне зображення (динамічний векторний метод).

Побудуємо на графічній поверхні контур квадрата у вигляді «рухомого вогника», як послідовність статичних пікселів, що вмикаються за вказаним на рис. 10б алгоритмом.

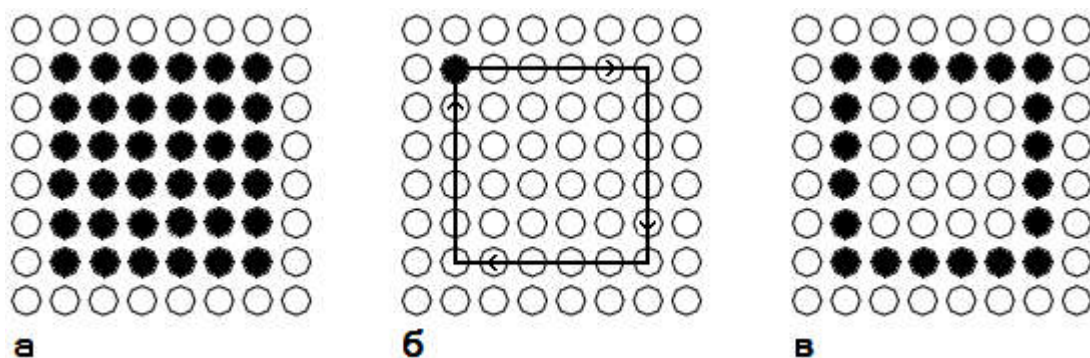


Рис. 10. Виведення зображення квадрата на світлодіодну матрицю

Для порівняння, залишимо у програмі можливість відображення заповненого квадрата (див. приклад 2) у окремому режимі, який вмикається за допомогою зовнішнього сигналу керування на 3-й лінії порту C.

// Приклад 3

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>
int i;

int main(void)
{
    DDRB = 0xFF;    // Port B - Out, Matrix - Row
    PORTB = 0x00;    // Clear Row
    DDRC = 0x00;    // Port C - In, Button
    PORTC = 0xFF;    // Port C - On pul-up
    DDRD = 0xFF;    // Port D - Out, Matrix - Col
    PORTD = 0x00;    // Clear Col

    while(1)
    {
        if (~PINC & (1<<2))    // 2 або PC2
        {
            PORTB = (1<<1);    // Row
            for (i=1; i<=6; i++)
                {PORTD = ~(1<<i); _delay_ms(800);}
            PORTD = ~(1<<6);    // Col
            for (i=1; i<=6; i++)
                {PORTB = (1<<i); _delay_ms(800);}
            PORTB = (1<<6);    // Row
            for (i=6; i>=1; i--)
                {PORTD = ~(1<<i); _delay_ms(800);}
            PORTD = ~(1<<1);    // Col
            for (i=6; i>=1; i--)
                {PORTB = (1<<i); _delay_ms(800);}
        }
        else
        {
            PORTB = 0b01111110;    // Row
            PORTD = 0b10000001;    // Col
        }
        _delay_ms(10);
    }
}
```


Для формування на графічній поверхні довільного зображення, воно попередньо розкладається на складові елементи в системі координат поверхні у відповідності з її роздільною здатністю – «розгортається» за стовпцями або рядками у вигляді кодових комбінацій, що визначають характеристики кожного окремого елемента (у нашому випадку – яскравості). З кодових комбінацій формується растровий масив, впорядкований за стовпцями або рядками. Надалі таке зображення може бути відтворено на графічній поверхні динамічним методом за допомогою зворотної операції (яка також називається розгорткою).

У даному випадку, для забезпечення однакового середнього струму на всіх пікселях, розгортка здійснюється за рядками бо через обмежувальні резистори під'єднані стовпці матриці (див. рис. 9). Побудуємо на графічній поверхні контур квадрата (див. рис. 10в) динамічним растровим методом.

// Приклад 4

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>

void preset()
{
    DDRB = 0xFF;    //Port B - Out, Matrix - Row
    PORTB = 0x00;
    DDRC = 0x00;    //Port C - In, Button
    PORTC = 0xFF;
    DDRD = 0xFF;    //Port D - Out, Matrix - Col
    PORTD = 0x00;
}

int Matrix[8]={ 0b00000000,
                0b01111110,
                0b01000010,
                0b01000010,
                0b01000010,
                0b01000010,
                0b01111110,
                0b00000000 };
```

```

int main(void)
{
    preset();
    while(1)
    {
        if (~PINC & (1 << PC2))
        {
            for (int i=0; i<=7; i++)
            {
                PORTB = 1 << i;
                PORTD = ~Matrix[i];
                _delay_ms(1);
            }
        }
        else
        {
            PORTB = (1 << 1);           // Row
            PORTD = ~0b01111110;       _delay_ms(50);
            PORTD = ~(1 << 6);          // Col
            PORTB = 0b01111110;        _delay_ms(50);
            PORTB = (1 << 6);           // Row
            PORTD = ~0b01111110;       _delay_ms(50);
            PORTD = ~(1 << 1);          // Col
            PORTB = 0b01111110;        _delay_ms(50);
        }
    }
}

```

Для порівняння, у другому режимі роботи програми тут відображується контур квадрата динамічним векторним методом.

Завдання

1. Вивести запропоноване викладачем зображення на світлодіодну матрицю 8x8 різними методами відтворення.

2. Випробувати декілька способів перемикання режимів відтворення.

Інформацію щодо роботи з окремими бітами регістрів засобами мови C наведено у дод. Е.

Контрольні запитання

1. Що таке растрова графіка?
2. Які параметри характеризують растрове зображення?
3. Що таке векторна графіка?
4. Які методи відтворення зображень вам відомі?
5. Що таке розгортка і які види розгорток вам відомі?

Розрахунково-графічна робота

КЕРУВАННЯ СВІТЛОДІОДНОЮ МАТРИЦЕЮ

Мета роботи: Закріпити відповідні знання та вміння роботи з мікропроцесорними контролером, набуті під час вивчення дисципліни.

Зміст та оформлення роботи

Розрахунково-графічна робота оформлюється у вигляді робочого проекту мікроконтролерної системи керування промисловим індикатором (світлодіодною матрицею 8x8). Робота повинна містити графічну частину та записку з наступними складовими:

1. постановка задачі;
2. аналіз задачі;
3. програма керування мовою C;
4. імітаційна схема для ISIS Proteus.

Графічна частина включає креслення блок-схеми алгоритму роботи системи керування (див. дод. К) та креслення схеми комутації мікроконтролера ATmega8 (у PDIP або TQFP корпусі [16]) з іншими електронними компонентами системи, виконані у графічному редакторі з дотриманням чинних вимог ЄСКД та ЄСПД [12 – 15].

Захист роботи відбувається шляхом випробування розробленої системи керування на емуляторі або макетній платі.

Зауваження 1

Слід відзначити, що імітаційна схема ISIS не є потрібним кресленням, бо виводи компонентів там згруповані за функціональною ознакою (а не у відповідності з їх форм-фактором) і система живлення у багатьох випадках не представлена взагалі.

Завдання

Розробити мікроконтролерну систему керування промисловим індикатором (світлодіодною матрицею 8x8), яка реалізує наступні режими роботи:

1. Виведення на індикатор шифру вашої навчальної групи *динамічним* методом.
2. Виведення на індикатор «рухомого вогника» *статичним* методом за алгоритмом, що відповідає вашому варіанту (див. дод. Ж).
3. Виконання *оригінального* (третього) режиму роботи індикатора, доступного у системі за допомогою *оригінального* способу перемикання режимів.

Зауваження 2

Наявність третього режиму у проєкті не обов'язкова але впливає на загальну оцінку. Виведення покадрової динамічної анімації на індикатор оригінальною роботою не вважається але також підвищує загальну оцінку.

Список рекомендованої літератури

1. Новацький А.О. Програмування мікроконтролерів родини AVR : Навчальний посібник. – Київ : КПІ, 2013. – 109 с.
2. Руководство пользователя программы ISIS Proteus VSM. / Радио-Ежегодник, 2013, Вып. 24., – 443 с.
3. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы Atmel. – М. : Додэка-XXI, 2008. – 560 с.
4. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR. – СПб. : Наука и Техника, 2008. – 544 с.
5. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel. – М. : РадиоСофт, 2002. – 176 с.
6. Хартов В.Я. Микроконтроллеры AVR. Практикум для начинающих. – М. : МГТУ им. Н.Э. Баумана, 2007. – 240 с.
7. Мортон Дж. Микроконтроллеры AVR. Вводный курс. – М. : Додэка-XXI, 2006. – 272 с.
8. Прокопенко В.С. Программирование микроконтроллеров Atmel на языке C. – Київ : МК-Пресс, 2012. – 320 с.
9. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М. : Додэка-XXI, 2004. – 288 с.
10. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному. – М. : СОЛОН-Пресс, 2003. – 288 с.
11. Безуглов Д.А., Калиенко И.В. Цифровые устройства и микропроцессоры : учебное пособие. – Ростов н/Д : Феникс, 2008. – 468 с.
12. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. – Київ : УкрНДНЦ, 2016. – 31 с.
13. Усатенко С.Т., Каченюк Т.К., Терехова М.В. Выполнение электрических схем по ЕСКД : Справочник. – М. : Издательство стандартов, 1989. – 325 с.
14. Р 50-77-88. Рекомендации ЕСКД. Правила выполнения диаграмм. / Единая система конструкторской документации. – М. : Издательство стандартов, 1989. – 9 с.
15. ДСТУ ISO 5807:2016 Обробляння інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT). – Київ : УкрНДНЦ, 2016. – 18 с.
16. 8-bit ATmega8 / Dataheet Rev. 2486AA–AVR–02/2013. – San Jose, CA : Atmel Corporation, 2013. – 331 с.

Властивості 8-розрядних AVR мікроконтролерів

Features	Mega	Tiny
Flash	4 – 256 KB	0.5 – 16 KB
SRAM	512 B – 16 KB	32 B – 1 KB
External	до 64 K	0
EEPROM	512 B – 4 KB	0 – 512 B
Interrupt	до 45	5 – 16
External	до 32	до 13
ADC 10 bit (channels)	8 – 16	4 – 28
DAC	0, 1	0
Comparators	1 – 4	1, 2
Prog Counter (bits)	11 – 17	9 – 14
Counters 8 bit	1, 2	1, 2
PWM (channels)	0 – 8	1, 2
Counters 16 bit	1 – 4	0, 1
PWM (channels)	2 – 12	2 – 8
I/O & Packages	28 – 100 pin	6 – 32 pin
Parallel Interface	23 – 86 pin	4 – 28 pin
Serial Interface		
SPI	1 – 5	1, 2
USART	0 – 4	0 – 2
TWI	0, 1	0, 1
USB	0, 1	0
Debug Interface		
JTAG	0, 1	0
debugWIRE	0	0, 1
Oper-g Voltage	1.8 – 6.0 V	1.8 – 6.0 V
Speed Grade	4 – 20 MHz	4 – 20 MHz

Other Peripheral Features для сімейства Mega	CAN (Control Area Network)	NAND Interface (SSD накопичувач)
	Video Decoder	Graphic LCD
	Camera Interface	Ethernet
	QTouch (сенсорні кнопки ємнісного типу)	LIN (Local Interconnect Network)
	Resistive Touch Screen	FPU (Floating Point Unit)
	MPU (Multiple Process Unit)	MMU (Memory Management Unit)
	Crypto Engine	Temperature Sensor
	External Bus Interface	DRAM Memory

Система команд мікроконтролера АТmega8

Команда	Операнд	Опис інструкції
<i>Арифметичні та логічні операції</i>		
ADD	Rd, Rr	Скласти (add) вміст регістрів
ADC	Rd, Rr	Скласти вміст регістрів з урахуванням біта C
ADIW	Rd1, K	Скласти безпосередньо (immediate) слово (word) у регістровій парі з константою K = 0 .. 63
SUB	Rd, Rr	Відняти (subtract) вміст регістрів
SUBI	Rd, K	Відняти безпосередньо константу K = 0 .. 255 від вмісту регістра
SBC	Rd, Rr	Відняти вміст регістрів з переносом C
SBCI	Rd, K	Відняти безпосередньо (immediate) константу K = 0 .. 255 від вмісту регістра з переносом C (d = 16 .. 31)
SBIW	Rd1, K	Відняти безпосередньо зі слова (word) у регістровій парі константу K = 0 .. 63
AND	Rd, Rr	Виконати побітову операцію and над вмістом регістрів
ANDI	Rd, K	Виконати безпосередньо побітову операцію and над вмістом регістра та константою K = 0 .. 255 (d = 16 .. 31)
OR	Rd, Rr	Виконати побітову операцію or над вмістом регістрів
ORI	Rd, K	Виконати безпосередньо побітову операцію or над вмістом регістра та константою K = 0 .. 255 (d = 16 .. 31)
EOR	Rd, Rr	Виконати побітову операцію xor над вмістом регістрів
COM	Rd	Виконати інверсію регістра (complement), зворотний код
NEG	Rd	Змінити знак вмісту регістра, доповнений код
INC	Rd	Інкремент (increment) вмісту регістра
DEC	Rd	Декремент (decrement) вмісту регістра
TST	Rd	Перевірити (test) на нуль або мінус вміст регістра
CLR	Rd	Очистити регістр (clear register)
SER	Rd	Встановити всі біти регістра (set register) (d = 16 .. 31)
MUL	Rd, Rr	Множення (multiply) чисел без знаку (unsigned) у регістрах
MULS	Rd, Rr	Множення чисел із знаком (signed) у регістрах
MULSU	Rd, Rr	Множення чисел із знаком (signed) і без знаку (unsigned) у регістрах
FMUL	Rd, Rr	Множення дробових (fractional) беззнакових (unsigned) чисел у регістрах

FMULS	Rd, Rr	Множення дробових (fractional) чисел із знаком (signed) у регістрах
FMULSU	Rd, Rr	Множення дробового числа із знаком (signed) та без знаку (unsigned) у регістрах
<i>Операції з бітами</i>		
SBI	P, b	Встановити біт з номером b у I/O-регістрі (set bit in i/o) (P = 0 .. 31, 63)
CBI	P, b	Очистити біт з номером b у I/O-регістрі (clear bit in i/o) (P = 0 .. 31, 63)
SBR	Rd, K	Встановити біти регістра за маскою K = 0 .. 255 (d = 16 .. 31)
CBR	Rd, K	Очистити біти регістра за маскою K = 0 .. 255 (d = 16 .. 31)
LSL	Rd	Логічний зсув ліворуч (logical shift left)
LSR	Rd	Логічний зсув праворуч (logical shift right)
ROL	Rd	Зсув ліворуч (rotate left) через біт C (carry)
ROR	Rd	Зсув праворуч (rotate right) через біт C (carry)
ASR	Rd	Арифметичний зсув праворуч (arithmetic shift right)
SWAP	Rd	Обміняти (swap) напівбайти регістра місцями
BSET	s	Встановити біт (bit set) з номером s у регістрі SREG
BCLR	s	Очистити біт (bit clear) з номером s у регістрі SREG
BST	Rr, b	У біт T записати вміст біта з номером b регістра (bit store)
BLD	Rd, b	У біт з номером b регістра записати біт T (bit load)
SE ...		Встановити ознаку (set)
SEC		Carry Flag
SEN		Negative Flag
SEZ		Zero Flag
SEI		Global Interrupt Enable
SES		Signed Test Flag
SEV		Two's Complement Overflow
SET		T-flag
SEH		Half Carry Flag
CL ...		Очистити ознаку (clear)
CLC		Carry Flag
CLN		Negative Flag
CLZ		Zero Flag
CLI		Global Interrupt Disable
CLS		Signed Test Flag
CLV		Two's Complement Overflow
CLT		T-flag

CLH		Half Carry Flag
<i>Транспортні операції</i>		
MOV	Rd, Rr	Скопіювати (move) вміст регістра в інший регістр
MOVW	Rd, Rr	Скопіювати вміст регістрової пари в іншу пару (word)
PUSH	Rr	Записати регістр у стекову пам'ять
POP	Rd	Відновити регістр зі стекової пам'яті
IN	Rd, P	Завантажити регістр з порту (P = 0 .. 63)
OUT	P, Rr	Записати в порт з регістра (P = 0 .. 63)
LDI	Rd, K	Завантажити регістр безпосередньо (immediate) константою K = 0 .. 255 (d = 16 .. 31)
LD	Rd, X	Завантажити регістр опосередковано (load) з комірки ОЗУ з адресою у індексному регістрі
	Rd, Y	
	Rd, Z	
	Rd, X+	Завантажити опосередковано з після-інкрементом (post-inc)
	Rd, Y+	
	Rd, Z+	
	Rd, -X	Завантажити опосередковано з перед-декрементом (pre-dec)
	Rd, -Y	
	Rd, -Z	
LDD	Rd, Y+q	Завантажити опосередковано із зміщенням q = 0 .. 63 (displacement)
	Rd, Z+q	
LDS	Rd, k	Завантажити регістр прямо (direct) з комірки пам'яті (SRAM) з адресою k = 0 .. 64 KB
ST	X, Rr	Зберегти (store) регістр опосередковано за адресою в індексному регістрі
	Y, Rr	
	Z, Rr	
	X+, Rr	Зберегти опосередковано з після-інкрементом (post-inc).
	Y+, Rr	
	Z+, Rr	
	-X, Rr	Зберегти опосередковано з перед-декрементом (pre-dec)
	-Y, Rr	
	-Z, Rr	
STD	Y+q, Rr	Зберегти опосередковано із зміщенням q = 0 .. 63 (displacement)
	Z+q, Rr	
STS	k, Rr	Зберегти прямо у SRAM за адресою k = 0 .. 65535
LPM		
	Rd, Z	Завантажити регістр з Program Memory опосередковано
	Rd, Z+	Завантажити регістр з Program Memory з після-інкрементом
SPM		Store Program Memory

<i>Операції керуванням виконанням програми</i>		
RJMP	k	Перейти відносно (relative) із зміщенням адреси $k = -2 \text{ KW} \dots 2 \text{ KW}$
IJMP		Перейти опосередковано (indirect), за адресою у Z
RCALL	k	Викликати підпрограму відносно (relative) із зміщенням адреси $k = -2 \text{ KW} \dots 2 \text{ KW}$
ICALL		Викликати підпрограму опосередковано (indirect), за адресою у Z
RET		Повернутися з підпрограми (return)
RETI		Повернутися з підпрограми переривання (interrupt return)
CPSE	Rd, Rr	Порівняти вміст регістрів і пропустити наступну команду (skip) якщо однаковий (equal)
CP	Rd, Rr	Порівняти вміст регістрів (compare)
CPC	Rd, Rr	Порівняти вміст регістрів з урахуванням переносу (carry)
CPI	Rd, K	Порівняти вміст регістра безпосередньо (immediate) з константою $K = 0 \dots 255$ ($d = 16 \dots 31$)
SB ...		Пропустити (skip) наступну команду, якщо біт (bit) з номером $b = 0 \dots 7$ у регістрі ...,
SBRC	Rr, b	очищено (register cleared)
SBRS	Rr, b	встановлено (register set)
SBIC	P, b	очищено (I/O-register cleared)
SBIS	P, b	встановлено (I/O-register set)
BR ...		Перейти (branch) на адресу із зміщенням $k = -64 \dots +63$, якщо: ...
BRBS	s, k	встановлено (set) біт з номером s регістра SREG
BRBC	s, k	очищено (cleared) біт з номером s регістра SREG
BREQ	k	дорівнює (equal)
BRNE	k	не дорівнює (not equal)
BRCS	k	встановлено ознаку переносу (carry set)
BRCC	k	очищено ознаку переносу (carry cleared)
BRSH	k	дорівнює або більше (same or higher) без знаку
BRLO	k	менше (lower), без знаку
BRMI	k	мінус (minus)
BRPL	k	плюс (plus)
BRGE	k	більше або дорівнює (greater or equal) з урахуванням знаку (signed)
BRLT	k	менше ніж нуль (less than zero) з урахуванням знаку
BRHS	k	встановлено ознаку напівпереносу (half-carry set)
BRHC	k	очищено ознаку напівпереносу (half-carry clear)

BRTS	k	ознаку T встановлено (set)
BRTC	k	ознаку T очищено (cleared)
BRVS	k	ознаку переповнення встановлено (overflow set)
BRVC	k	ознаку переповнення очищено (overflow cleared)
BRIE	k	глобальне переривання дозволено (interrupt enabled)
BRID	k	глобальне переривання заборонено (interrupt disabled)
<i>Операції керування MCU</i>		
NOP		No Operation
SLEEP		Увімкнути режим енергозбереження
WDR		Watchdog Reset

У таблиці прийнято наступні позначення:

Позначення	Зміст
Rd	GPR регістр Destination, d = (0 .. 31)
Rr	GPR регістр souRce, r = (0 .. 31)
Rdl	одна з 4-х регістрових пар Rdl:Rdh , dl = (24, 26, 28, 30)
K	константа (дані 6 або 8 розрядів)
k	зміщення $\pm 2K$ адреси у PC або пряма адреса
P	адреса порту (порти: перші 64 I/O-регістри)
b	номер біта регістра, b = (0 .. 7)
Rr(b)	розряд GPR регістра Rr
P(b)	розряд порту P (I/O-регістра)
s	номер біта регістра SREG , s = (0 .. 7), див. рис. Д1
q	6-розрядне зміщення при індексній адресації
X, Y, Z	індексні регістри R26:R27 , R28:R29 , R30:R31

Розташування бітів ознак регістра статусу **SREG** зображено на рис. Д1.

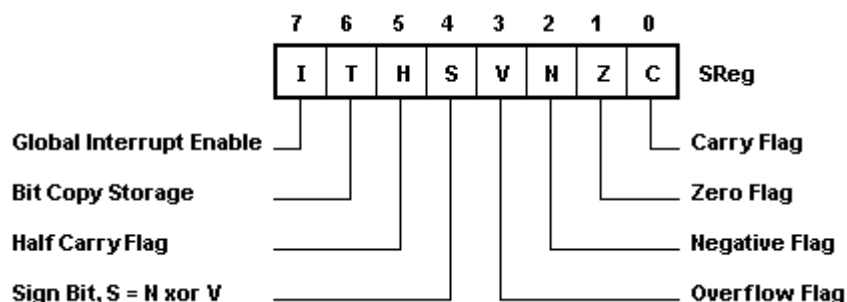


Рис. Д1. Ознаковий регістр AVR контролера

Службові регістри мікроконтролера АТmega8

Адреса	Ім'я	Призначення
0x3F	SREG	Status Register
0x3E	SPH	Stack Pointer
0x3D	SPL	
0x3B	GICR	General Interrupt Control Register
0x3A	GIFR	General Interrupt Flag Register
0x39	TIMSK	Timer Interrupt Mask Register
0x38	TIFR	Timer Interrupt Flag Register
0x37	SPMCR	Store Program memory Control Register
0x35	MCUCR	MCU Control Register
0x34	MCUCSR	MCU Control and Status Register
0x33	TCCR0	Timer/Counter 0 Control Register
0x32	TCNT0	Timer/Counter 0 Register
0x31	OSCCAL	Oscillator Calibration Register
0x30	SFIOR	Special Function IO Register
0x2F	TCCR1A	Timer/Counter 1 Control Registers
0x2E	TCCR1B	
0x2D	TCNT1H	Timer/Counter 1 Register
0x2C	TCNT1L	
0x2B	OCR1AH	Output Compare Register 1A
0x2A	OCR1AL	
0x29	OCR1BH	Output Compare Register 1B
0x28	OCR1BL	
0x27	ICR1H	Input Capture Registers 1
0x26	ICR1L	
0x25	TCCR2	Timer/Counter 2 Control Register
0x24	TCNT2	Timer/Counter 2 Register
0x23	OCR2	Output Compare Register 2
0x22	ASSR	Asynchronous Status Register
0x21	WDTCSR	Watchdog Timer Control Register
0x1F	EEARH	EEPROM Address Register
0x1E	EEARL	
0x1D	EEDR	EEPROM Data Register
0x1C	EECR	EEPROM Control Register
0x18	PORTB	GPIO Port B Data Register
0x17	DDRB	GPIO Port B Data Direction Register
0x16	PINB	GPIO Port B Input Pins Address

0x15	PORTC	GPIO Port C Data Register
0x14	DDRC	GPIO Port C Data Direction Register
0x13	PINC	GPIO Port C Input Pins Address
0x12	PORTD	GPIO Port D Data Register
0x11	DDRD	GPIO Port D Data Direction Register
0x10	PIND	GPIO Port D Input Pins Address
0x0F	SPDR	SPI Data Register
0x0E	SPSR	SPI Status Register
0x0D	SPCR	SPI Control Register
0x0C	UDR	USART I/O Data Register
0x0B	UCSRA	USART Control and Status Registers
0x0A	UCSRB	
0x20	UCSRC	
	UBRRH	USART Baud Rate Register
0x09	UBRRL	
0x08	ACSR	Analog Comparator Control and Status Register
0x07	ADMUX	ADC Multiplexer Selection Register
0x06	ADCSRA	ADC Control and Status Register
0x05	ADCH	ADC Data Register
0x04	ADCL	
0x36	TWCR	TWI Control Register
0x03	TWDR	TWI Data Register
0x02	TWAR	TWI (Slave) Address Register
0x01	TWSR	TWI Status Register
0x00	TWBR	TWI Bit Rate Register

У таблиці адреси регістрів вказані для ізольованої області введення/виведення. Для визначення адреси регістра у загальному просторі пам'яті даних контролера треба до вказаної адреси додати зміщення 0x20.

Паралельний порт AVR-контролера

Кожен з портів введення/виведення загального призначення GPIO (які у термінології Atmel також називаються General Digital I/O Ports) містить три 8-бітні регістри, зображені на рис. Д2.

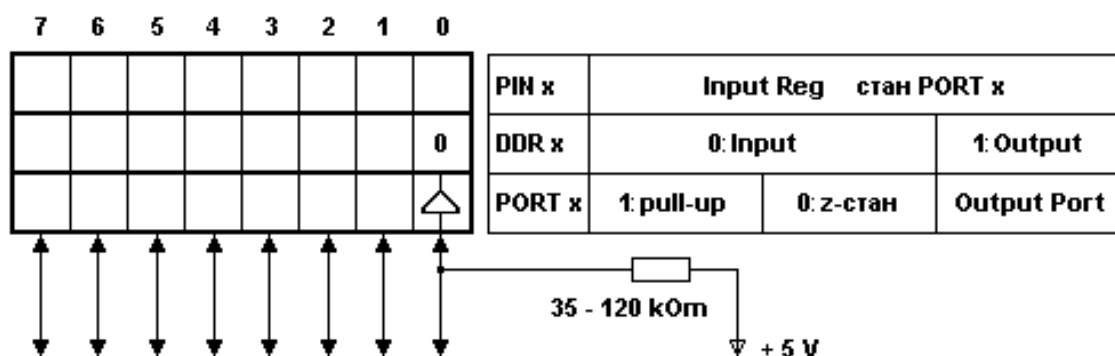


Рис. Д2. Регістри GPIO портів

Напрямок роботи кожної лінії I/O, що утворюють порт, визначається вмістом відповідного біта регістра **DDR_x** (де x – ім'я порту, наприклад **D**). У режимі Output інформація для виведенні назовні береться з бітів регістра **PORT_x**.

У режимі Input інформація зчитується з регістра **PIN_x** (який відображує фізичний стан лінії I/O). При цьому значення бітів регістра **PORT_x** визначає, чи під'єднано вбудований pull-up резистор до відповідної лінії I/O. Якщо pull-up резистор не задіяно, ввід мікросхеми перебуває у високоімпедансному стані (z-стан, див. роботу 3).

Таким чином, при введенні через відповідну лінію I/O логічного нуля за допомогою комутуючого пристрою можна використовувати як вбудований, так і зовнішній pull-up резистор. При введенні логічної одиниці слід використовувати лише зовнішній pull-down резистор на високоімпедансному виводі мікросхеми.

Формат файлу прошивки AVR-контролера

Всі сучасні утиліти для запису інформації у мікросхеми з пам'яттю (EEPROM, мікроконтролери тощо) підтримують формат даних Intel HEX (1973, розширення *.hex), призначений для представлення двійкової інформації (8-, 16- або 32-бітної) у текстовій формі (символами ASCII) і ставший фактичним стандартом зберігання об'єктного модуля прошивки (firmware).

Окремий рядок 8-бітного hex-файлу для AVR-контролера містить запис з наступними полями:

Поле	Зміст	Розмір
:	Тег початку рядка	1 символ
RecLen	Розмір поля Data	1 байт (2 символи)
OffSet	16-бітне зміщення адреси	2 байти
RecType	Тип запису (00, 01)	1 байт
Data	Дані (або адреса)	RecLen байт
Checksum	Контрольна сума	1 байт

Кожен байт запису (крім тега початку) подається у шістнадцятковій формі за допомогою двох текстових символів. Таким чином, максимальна кількість байтів у полі **Data** одного рядка не може перевищувати 255 (або 510 символів), хоча, зазвичай, там розміщують до 16 байтів (32 символи).

Рядок запису закінчується стандартними символами CR/LF (коди 0D та 0A), які у текстових редакторах зазвичай не відображуються.

Пакет корисних даних (**RecType** = 00) записується у адресний простір пристрою послідовно із зміщенням **OffSet**, починаючи з абсолютної адреси 0x0000 або з адреси початку сектора завантажувача (bootloader, див. таб. 1).

Checksum розраховується, як сума значень байтів всіх інших полів за модулем 0d256 і подається як від'ємне число у доповненому коді. Таким чином, сума значень всіх байтів рядка повинна дорівнювати 0.

Для даних всіх типів крім **RecType** = 00, поле зміщення **Offset** задається як 0000.

Останній рядок hex-модуля завжди містить маркер завершення файлу (**RecType** = 01), та має характерний вигляд (: 00 0000 01 FF).

16- та 32-бітні hex-файли забезпечують більшу кількість способів адресації даних і для цього використовують більшу кількість типів записів (**RecType** = 00 ... 05).

Наприклад, рядок (: 02 0000 03 0100 FA) встановлює початкову абсолютну адресу 0x0100, від якої будуть відраховуватись потрібні зміщеннями **Offset** для запису наступних пакетів даних (початок сегмента, **RecType** = 03). Інколи у hex-файлі для AVR-контролера також зустрічається запис 03 типу, але такий рядок при програмуванні ігнорується.

Файл об'єктного модуля у форматі *.elf (Executable and Linkable Format) є двійковим і крім основної програми може містити додаткові розділи, наприклад, для програмування fuse-регістрів, енергонезалежної пам'яті EEPROM тощо.

Операції з бітами у мові С

Мова асемблера для AVR-контролера (див. дод. Б) містить команди, що забезпечують прямий доступ до окремих бітів регістрів ізольованої області введення / виведення (портів **P**). У мові С можливості роботи з окремими бітами числа обмежені, але потрібного ефекту можна домогтися (як і у будь-якій іншій мові програмування) за допомогою операцій маскуванню [8].

Операції маскуванню засновані на використанні елементарних логічних функцій а операції імплікації крім того використовують і оператори порівняння / відношення, які також повертають логічний результат.

У мові С самостійний логічний тип даних відсутній і вказані операції реалізуються через цілі числові типи, де **0** відповідає *хибі* а будь-яке інше значення – *істині* (хоча у мові С++ визначено скалярний тип **bool** на множині значень **1** – **true** та **0** – **false**).

Побітові операції виконуються над цілими двійковими числами, які у програмі можуть бути подані у різних системах числення за допомогою відповідного префікса (літерала), наприклад, десяткове число 255 можна записати, як:

- двійкове **0b11111111;**
- шістнадцяткове **0xFF;**
- вісімкове **0377;**
- десяткове **0d255** або просто **255.**

У термінології Atmel шістнадцяткові числа також використовуються у ситаксисі мови Pascal, наприклад, **\$FF** (на відміну від **FFh** у Intel-асемблері).

Як і арифметичні, побітові бінарні операції у мові C підтримують складений синтаксис (з оператором присвоєння).

Наприклад, запис `a += b` еквівалентний `a = a + b`.

Оператор	Синтаксис	Приклад
Присвоєння (assignment)	<code>a = b</code>	<code>a = 0b0011; n = 1; b = 0b1010; false = 0</code>
<i>Операції порівняння/відношення (comparison/relational)</i>		
Дорівнює (equal)	<code>a == b</code>	<code>(a == b) == false</code>
Не дорівнює (not equal)	<code>a != b</code>	<code>(a != b) == !false</code>
Більше (greater than)	<code>a > b</code>	<code>(a > b) == false</code>
Менше (less than)	<code>a < b</code>	<code>(a < b) == !false</code>
Більше або дорівнює (... or equal)	<code>a >= b</code>	<code>(a >= b) == false</code>
Менше або дорівнює (... or equal)	<code>a <= b</code>	<code>(a <= b) == !false</code>
<i>Логічні операції (logical)</i>		
Логічне заперечення НЕ (NOT)	<code>!a</code>	<code>(!a) == !false</code>
Логічне І (AND)	<code>a && b</code>	<code>(a && b) == false</code>
Логічне АБО (OR)	<code>a b</code>	<code>(a b) == false</code>
<i>Побітові операції (bitwise)</i>		
Побітове заперечення НЕ (NOT)	<code>~a</code>	<code>(~a) == 0b1100</code>
Побітове І (AND)	<code>a & b</code>	<code>(a & b) == 0b0010</code>
Побітове АБО (OR)	<code>a b</code>	<code>(a b) == 0b1011</code>
Побітове Виключне АБО (XOR)	<code>a ^ b</code>	<code>(a ^ b) == 0b1000</code>
Побітовий зсув ліворуч (left shift)	<code>a << n</code>	<code>(a << n) == 0b0110</code>
Побітовий зсув праворуч (right ...)	<code>a >> n</code>	<code>(a >> n) == 0b0001</code>

Операція маскування дозволяє аналізувати вміст окремих бітів двійкового слова шляхом занесення у інші біти наперед відомої комбінації цифр (зазвичай лише `0` або лише `1`), яку легко врахувати при подальшій перевірці вмісту всього слова (імплікації).

Той самий підхід використовується при маніпуляціях з окремими бітами слова, коли треба у конкретний розряд встановити (set) `1`, скинути (clear) `1` (або встановити `0`) чи інвертувати його вміст (toggle).

У всіх випадках `0` у відповідні розряди слова встановлюється побітовою операцією AND, `1` – побітовою операцією OR а інвертування – побітовою операцією XOR.

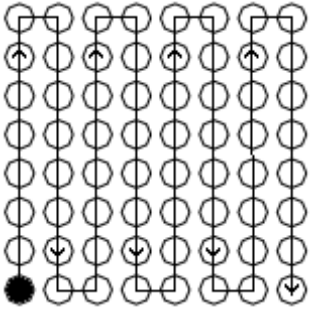
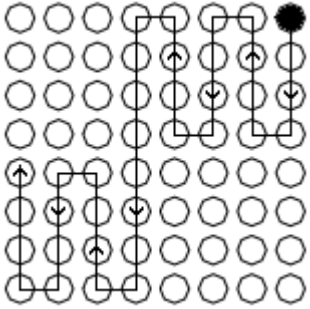
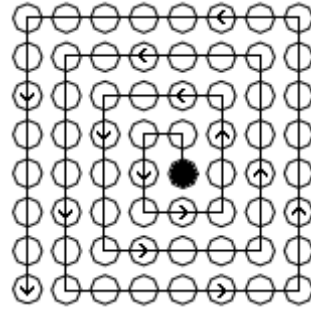
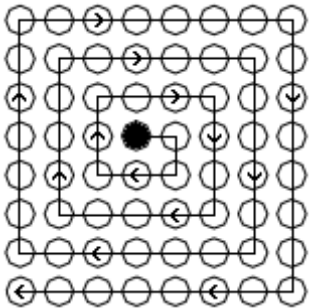
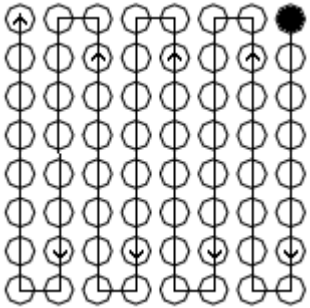
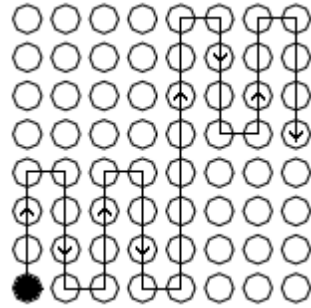
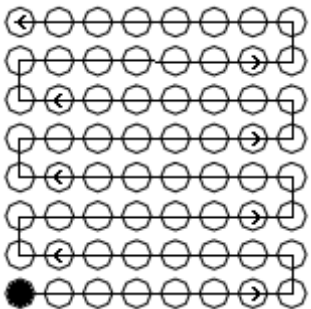
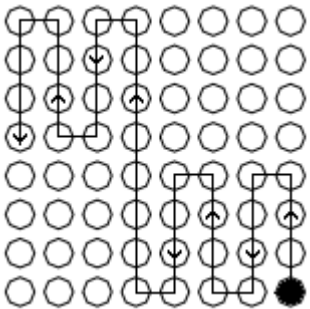
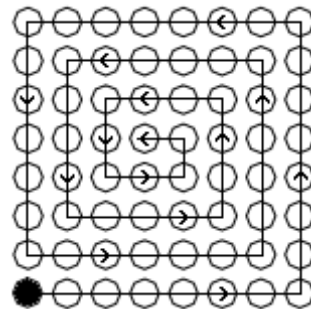
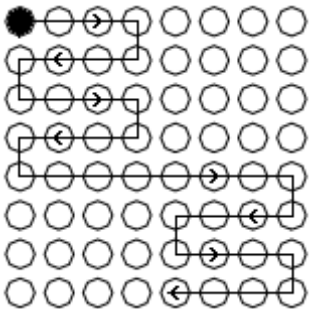
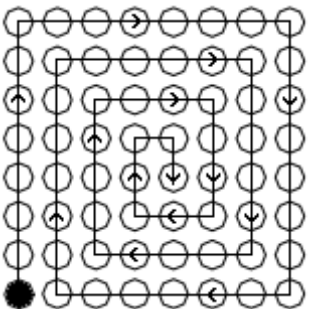
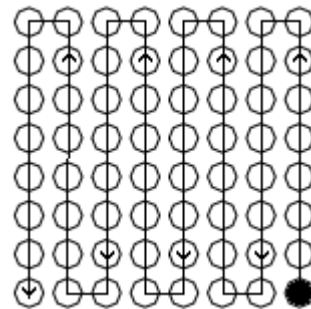
Операція	Приклад
Формування маски нулів з 1 у 0 -му біті	<code>mask = 0b00000001;</code> або <code>mask = 1;</code> або <code>mask = (1 << 0);</code>
Формування маски нулів з 1 у n -му біті	<code>mask = 0b00000100;</code> де <code>n = 2;</code> або <code>mask = (1 << 2);</code>
Формування маски нулів з декількома 1	<code>mask = 0b00100101;</code> або <code>mask = (1 << 5) (1 << 2) (1 << 0);</code>
Формування маски одиниць з 0 у n -му біті	<code>mask = 0b11111101;</code> де <code>n = 1;</code> або <code>mask = ~(1 << 1);</code>
Формування маски одиниць з декількома 0	<code>mask = 0b00100101;</code> <code>~mask == 0b11011010;</code>
Накладання маски нулів	<code>reg = 0bxxxxxxxx;</code> <code>mask = 0b00100101;</code> <code>reg &= mask;</code> або <code>reg = reg & mask;</code> <code>reg == 0b00x00x0x</code>
Накладання маски одиниць	<code>reg = 0bxxxxxxxx;</code> <code>mask = 0b00100101;</code> <code>reg = mask;</code> або <code>reg = reg mask;</code> <code>reg == 0bxx1xx1x1</code>
Встановлення 0 (clear) у n -й біт регістра	<code>reg = 0bxxxxxxxx;</code> де <code>n = 3;</code> <code>reg &= ~(1<<n);</code> або <code>reg = reg & ~0x08;</code> <code>reg == 0bxxxx0xxx</code>
Встановлення 1 (set) у n -й біт регістра	<code>reg = 0bxxxxxxxx;</code> де <code>n = 2;</code> <code>reg = (1<<n);</code> або <code>reg = reg 0x04;</code> <code>reg == 0bxxxxx1xx</code>
Інверсія n -го біта регістра	<code>reg ^= (1 << n);</code>
Інверсія (toggle) бітів регістра за маскою	<code>mask = 0b00100101;</code> інверсія бітів 0, 2 та 5 <code>reg ^= mask;</code>

Особливості мови C роблять зручнішим застосування логічної операції імплікації (implication) у постановці: якщо *посилка* операції (antecedent) повертає не **0**, виконується *наслідок* (consequent).

Операція	Приклад
Перевірка n -го біта регістра на 0	<code>(port & (1<<n) == 0)</code> або <code>!(port & (1<<n))</code> або <code>~(port & (1<<n))</code> або <code>(~port & (1<<n))</code>
Перевірка n -го біта регістра на 1	<code>(port & (1<<n) != 0)</code> або <code>(port & (1<<n))</code>
Перевірка рівності вмісту регістрів	<code>(port == reg)</code> або <code>!(port ^ reg)</code> або <code>~(port ^ reg)</code>

Завдання для програмування мікроконтролера

Нумерація рядків та стовпців світлодіодної матриці 8x8 відповідає вказаній на рис. 5.

Вар.	Завдання	Вар.	Завдання	Вар.	Завдання
1		2		3	
4		5		6	
7		8		9	
10		11		12	

Вар.	Завдання	Вар.	Завдання	Вар.	Завдання
13		14		15	
16		17		18	
19		20		21	
22		23		24	
25		26		27	

Вар.	Завдання	Вар.	Завдання	Вар.	Завдання
28		29		30	
31		32		33	
34		35		36	
37		38		39	
40		41		42	

Приклад виконання блок-схеми алгоритму керування

Креслення розробляється у відповідності із стандартом [15], аналогом міжнародного стандарту ISO 5807:1985. Нижче наведено один з варіантів блок-схеми алгоритму програми з прикладу 4 роботи 5, виконаний у графічному редакторі MS Visio (який містить бібліотеки готових примітивів для конструювання діаграм різних типів).

