МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «Прикарпатський національний університет
імені Василя Стефаника»

Фізико-технічний факультет

Голота В. І.

Методичні вказівки до виконання лабораторних робіт з дисципліни

“ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ”

Івано-Франківськ, 2022

УДК 004.42(076.5)
Г1

Рекомендовано до друку Вченою радою фізико-технічного факультету Прикарпатського національного університету імені Василя Стефаника (протокол № 3 від 27 жовтня 2022 року)

Рецензенти:

Когут Ігор Тимофійович, професор, завідувач кафедри комп'ютерної інженерії та електроніки Прикарпатського національного університету імені Василя Стефаника, доктор технічних наук

Яремій Іван Петрович, професор кафедри матеріалознавства і новітніх технологій Прикарпатського національного університету імені Василя Стефаника, доктор фізико-математичних наук

Г1 Голота В.І. Методичні вказівки до виконання лабораторних робіт з дисципліни "Програмування мікроконтролерів": [Електронний ресурс] / *Голота Віктор Іванович* / Фізико-технічний факультет; Прикарпатський національний університет імені Василя Стефаника. – Електронні текстові дані. – Івано-Франківськ, 2022. – 227 с.

Методичні вказівки до виконання лабораторних робіт розроблено з метою вивчення архітектури мікроконтролерів, структури і можливостей основних функціональних блоків, особливостей програмування 8-розрядних мікроконтролерів родини AVR корпорації Microchip. Викладений матеріал відповідає навчальним програмам з дисципліни "Програмування мікроконтролерів".

Методичні вказівки до виконання лабораторних робіт з дисципліни "Програмування мікроконтролерів" призначені для студентів галузі знань 12 "Інформаційні технології" спеціальності 123 "Комп'ютерна інженерія" та для студентів галузі знань 17 "Електроніка та телекомунікації" спеціальності 171 "Електроніка".

УДК 004.42(076.5)

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 1. Апаратно-програмне забезпечення мікроконтролерів

Мета роботи: вивчення апаратно-програмного забезпечення 8-бітових мікроконтролерів .

1. Архітектура мікроконтролерів

Мікроконтролер (МК) – це комп'ютер зі своїм обчислювальним пристроєм, постійною і динамічною пам’яттю, портами введення-виведення і різною периферією. Умовна схема МК AVR показана на рис. 1.

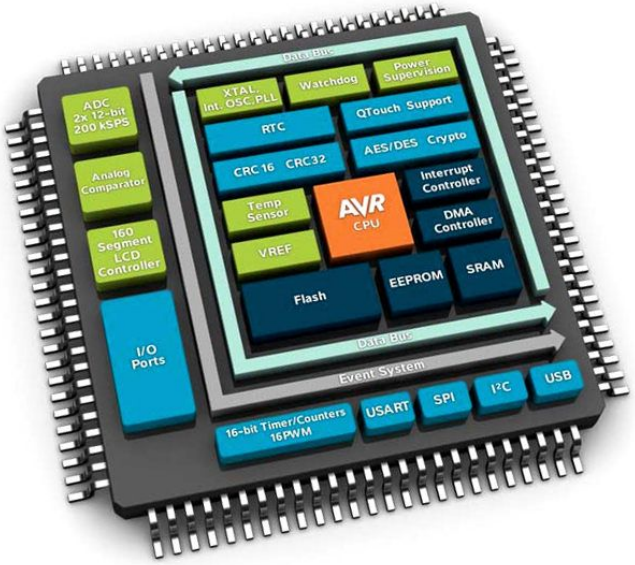


Рисунок 1 – Умовна схема МК AVR

Усередині МК містить:

– швидкодіючий процесор з RISC- архітектурою;

– FLASH-пам’ять;

– EEPROM- пам’ять;

– оперативну пам’ять SRAM;

– порти введення/виведення;

– периферійні і інтерфейсні модулі.

RISC (Reduced Instruction Set Computer) – архітектура з ретельно підібраним набором команд, які як правило виконуються за один такт роботи процесора. Сучасні МК Atmel містять близько 130 команд, які дуже швидко виконуються і не вимагають великих витрат як з процесорних ресурсів, так і з споживаної потужності.

Структурна схема МК Atmel зображена на рис. 2 і містить наступні функціональні блоки:

– **JTAG Interface** (Joint Test Action Group Interface) - інтерфейс внутрішньосхемного налагодження (4 дроти);

– **FLASH** - перепрограмовувана енергонезалежна пам’ять для зберігання програми;

– **Serial Peripheral Interface, SPI** - послідовний периферійний інтерфейс (3 дроти);

– **EEPROM** (Electrically Erasable Programmable Read-Only Memory) – перепрограмовувана енергонезалежна постійна пам’ять даних;

– **CPU** (ЦПП) - центральний процесорний пристрій, 8-бітове мікропроцесорне ядро;

– **ALU** (АЛП) - арифметико-логічний пристрій, основа блоку CPU;

– **RAM** (Random Access Memory) - оперативна пам’ять процесора з довільним доступом;

– **Program Counter** - лічильник команд;

– **32 General Purpose Registers** - 32 регістри загального призначення;

– **Instruction Register** - регістр команд, інструкцій;

– **Instruction Decoder** - декодер команд;

– **OCD** (On-Chip Debugger) - блок внутрішнього налагодження;

– **Analog Comparator** - аналоговий компаратор, блок порівняння аналогових сигналів;

– **A/D Converter** (Analog/Digital converter) - аналого-цифровий перетворювач;

– **LCD Interface** (Liquid-Crystal Display Interface) - інтерфейс для підключення рідко-кристалічного дисплея, індикатора;

– **USART** (Universal Asynchronous Receiver-Transmitter), UART - універсальний асинхронний приймач/передавач;

– **TWI** (Two-Wire serial Interface) - послідовний інтерфейс з дводротовим підключенням;

– **Watchdog Timer** - сторожовий або контрольний таймер;

– **I/O Ports** - порти введення/виведення;

– **Interrupts** - блок керування і реакції на переривання;

– **Timers/Counters** - модулі таймерів і лічильників.

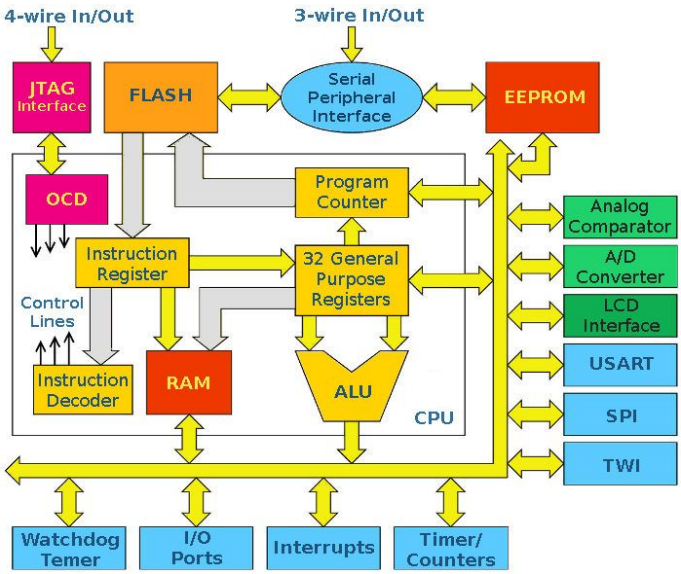


Рисунок 2 – Структурна схема МК AVR

Внутрішні блоки МК мають наступне призначення:

JTAG Interface – інтерфейс для внутрішнього налагодження безпосередньо в кристалі з використанням блоку внутрішнього налагодження (**OCD**) без використання емуляторів (для апаратного налагодження). Через JTAG-адаптер мікросхема напряму підключається до програмного комплексу для програмування і налагодження. Використовуючи даний інтерфейс можна в покроковому режимі виконувати програму прямо в МК, дивитися як змінюється вміст регістрів, як блимають індикатори і світлодіоди, які підключені до МК після кожного кроку і т. п. Для підключення до JTAG інтерфейсу достатньо 4-х провідників: TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), TMS (Test Mode Select). JTAG інтерфейс є у моделях МК, які мають більше 40 виводів і обсяг пам’яті більше 16 КБ.

FLASH – пам’ять програм, енергонезалежний ПЗП (постійний запам’ятовуючий пристрій), який виконаний за FLASH технологією. В пам’яті зберігається програма, яка буде

виконуватися блоком ALU МК. Флеш-пам’ять можна багатократно перезаписувати, тим самим змінюючи або доповнюючи програмний код для виконання. Даний тип пам’яті може зберігати записані в неї дані до 40 років, а кількість можливих циклів стирання/записування становить 10_000. В залежності від моделі МК розмір FLASH-пам’яті може досягати 256 кБ.

Serial Peripheral Interface, SPI – послідовний периферійний інтерфейс (SPI), який застосовується для обміну даними між декількома МК із швидкістю до декількох МГц. Для обміну даними по SPI інтерфейсу між двома пристроями достатньо 3-х провідників:

– MOSI (Master Output Slave Input) - дані від ведучого до веденого;

– MISO (Master Input Slave Output) - дані від веденого до ведучого;

– CLK (Clock) - тактовий сигнал.

Пристрої з SPI-інтерфейсом діляться на два типи: ведучий (Master) і ведений (Slave). Якщо до інтерфейсу підключено декілька пристроїв то для обміну даними між ними потрібні додаткові лінії зв’язку (провідники), щоб майстер міг вибрати ведений пристрій і зробити до нього запит. Також SPI інтерфейс використовується для внутрішньосхемного SPI програмування, по цьому інтерфейсу до МК підключається програматор.

EEPROM – енергонезалежна пам’ять даних в якій дані зберігатимуться навіть при відключенні живлення МК. У цій пам'яті можна зберігати налаштування для виконання програми, зібрані дані про статистику роботи пристрою і іншу корисну інформацію. Наприклад, зібравши невелику метеостанцію на МК, в EEPROM для кожного дня можна записати дані про температуру повітря, тиск, силу вітру, а потім у будь-який момент зчитати ці дані і провести статистичні дослідження. Для EEPROM виділений окремий адресний простір, який відрізняється від адресного простору RAM і FLASH. Пам'ять EEPROM МК – дуже цінний ресурс, оскільки її як правило дуже мало – 0,5÷1 кБ. Кількість перезаписів для цього типу пам'яті складає близько 100_000.

ALU – арифметико-логічний пристрій (АЛП), який синхронізований з тактовим сигналом і використовуючи стан лічильника команд (**Program Counter**) вибирає з пам’яті програм (**FLASH**) чергову команду і її виконує. Тактовий сигнал для МК задається тактовим генератором, і може бути поданий з декількох доступних джерел на вибір:

– внутрішній RC-генератор, який можна калібрувати на потрібну частоту;

– зовнішній керамічний або кварцовий резонатор з конденсаторами (не у всіх моделях);

– зовнішній тактовий сигнал.

Встановлюється джерело тактових імпульсів за допомогою FUSE-бітів. **FUSES (**з англ.: плавлення, пробка, запобіжник) – спеціальні 4 байти (4*8=32 біт) даних, які налаштовують деякі глобальні параметри МК в процесі прошивки.

Конфігурація бітів вказує МК:

– який вибрати задаючий генератор (зовнішній або внутрішній);

– ділити частоту генератора на коефіцієнт чи ні;

– використовувати вивід RESET для скидання або ж як додатковий вивід введення-виведення;

– кількість пам'яті для завантажувача;

– інші налаштування залежно від використовуваного МК.

CPU – це ядро МК, яке містить в АЛП, регістри і оперативну пам'ять. До АЛП підключений блок з 32-х регістрів загального призначення (**32 General Purpose Registers**-регістрова пам'ять) розміром 1 байт. Адресний простір регістрів загального призначення розміщений на початку оперативної пам’яті (SRAM), але не є її частиною. З даними, що поміщаються в регістри можна робити різноманітні арифметичні, логічні і бітові операції.

Виконати ці операції тільки в оперативній пам’яті неможливо. Для роботи з даними в RAM треба їх записати в регістри, зробити в регістрах потрібні операції, а потім записати результуючі дані з регістрів в пам’ять або в інші регістри.

SRAM – це статична оперативна пам’ять. У неї можна записувати дані з регістрів**,** зчитувати дані в регістри. Усі операції з даними виконуються в регістрах. Для різних родин МК Atmel розмір оперативної пам'яті різний і обмежений:

– ATxmega - до 32 кБ;

– ATmega - 16 кБ;

– ATtiny - 1 кБ.

Analog Comparator – цей блок порівнює між собою два рівні сигналу і запам'ятовує результат порівняння в певному регістрі, після чого результат можна проаналізувати і виконати необхідні дії. Для прикладу: можна використати цей блок як АЦП (аналого-цифровий перетворювач) і вимірювати напругу батареї живлення, у випадку якщо напруга батареї досягла низького рівня – виконати деякі дії, наприклад, засвітити червоний світлодіод. Також цей модуль можна застосовувати для виміру тривалості аналогових сигналів, зчитування встановлених режимів роботи пристрою за допомогою потенціометра і тому подібне.

A/D Converter – цей блок перетворює аналогове значення напруги в цифрове значення з яким можна працювати в програмі і на основі якого можна виконувати певні дії. Як правило діапазон напруги що подається на вхід АЦП в МК Atmel знаходиться в межах 0÷5,5 Вольт. Для цього блоку дуже важливо щоб МК живився від стабільного і якісного джерела живлення. У багатьох МК Atmel є спеціальний окремий вивід для подання стабільного живлення на схему АЦП.

LCD Interface – інтерфейс для підключення рідкокристалічного індикатора або дисплея. Застосуються для відображення інформації, стану пристрою і його вузлів.

USART – послідовний асинхронний інтерфейс для обміну даними з іншими пристроями. Підтримує протокол RS-232, завдяки чому МК можна під’єднати для обміну даними з комп’ютером. Для такого зв’язку МК з COM-портом комп’ютера потрібний конвертер логічних рівнів напруги (+12В для COM в +5В для МК), або ж просто RS232-TTL. Для цього використовують мікросхеми MAX232 і подібні до них. Для підключення МК до комп’ютера через USB по UART-інтерфейсу, можна використати спеціалізовану мікросхему FT232RL. Таким чином на нових комп'ютерах і ноутбуках де відсутній COM-порт можна під’єднати МК використовуючи USB-порт через USART інтерфейс.

TWI – інтерфейс для обміну даними по двопровідній шині. До такої шини даних можна підключити до 128 різних пристроїв, використовуючи дві лінії: тактовий сигнал (SCL) і сигнал даних (SDA). Інтерфейс TWI є аналогом базової версії інтерфейсу I2C. На відміну від SPI інтерфейсу (один майстер і один/декілька ведених) інтерфейс TWI – двонаправлений, що дозволяє організувати між декількома МК невелику внутрішню мережу.

Watchdog Timer – система контролю зависання МКз подальшим його перезапуском.

I/O Ports,**GPIO** – це набір блоків портів введення/виведення до виводів яких можна підключити різноманітні давачі. Кількість виводів портів в МК, може бути від 3 до 86. Вихідні драйвери в портах МК дозволяють безпосередньо підключати навантаження із струмом споживання 20 мА (максимум 40 мА) при напрузі живлення 5 В. Загальний струм навантаження для одного порту не повинен перевищувати значення в 80 мА (наприклад до 4 виводів одного з портів підключити світлодіоди із струмом 15-20 мА).

Interrupts – блок який відповідає за реакцію і запуск на виконання певних функцій при поданні сигналу на певні входи МК або ж по якійсь внутрішній події (наприклад тактування

таймера). Під кожне переривання розробляється і записується в пам'ять окрема підпрограма. При виникненні визначеної для переривання події основна програма **переривається** і пріоритетно виконується підпрограма, яка написана для оброблення поточного переривання. Після завершення виконання підпрограми оброблення переривання відбувається повернення до виконання основної програми у те місце де вона була перервана.

Timers/Counters – набір таймерів і лічильників. МК AVR мають від одного до чотирьох таймерів і лічильників. Вони використовуються для підрахунку кількості зовнішніх подій, формування сигналів певної тривалості, вироблення запитів на переривання і тому подібне. Розрядність таймерів і лічильників – 8 і 16 біт.

2. Апаратне забезпечення для вивчення МК AVR

Апаратне забезпечення яке потрібне для вивчення МК AVR:
- мікроконтролер ATmega 8515 або ATmega8 в корпусі DIP-28;
- макетна панель для безпаяльного монтажу (breadboard) і з'єднувальні провідники до неї;
- програматор, у даному випадку це USBASP;
- декілька світлодіодів різного кольору (Іном = 1÷5 мА);
- резистори 620÷800 Ом (по одному на кожний світлодіод);
- джерело живлення 5В;
- комп'ютер із встановленою ОС Linux (Debian, Ubuntu, Mint, Suse) або Windows.

Макетна панель, провідники, мікроконтролери і програматор USB ASP показані на рис. 3

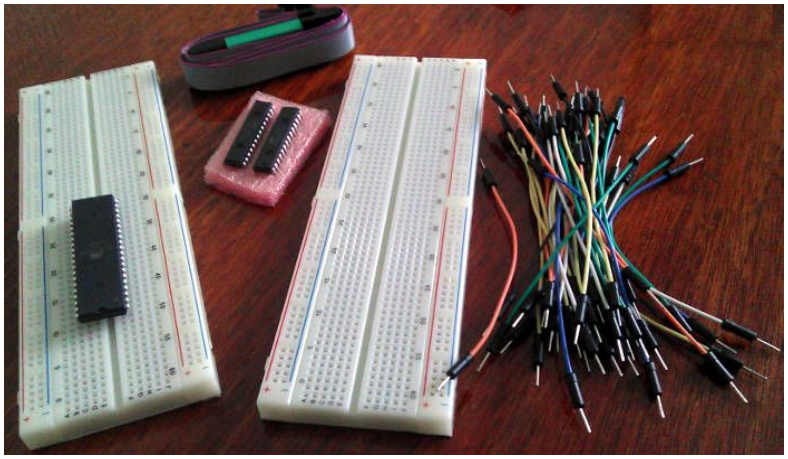


Рисунок 3 – Макетна панель, провідники, мікроконтролери і програматор USB ASP

Звичайно до МК підключають зовнішній кварц і конденсатори, які задають частоту внутрішнього тактового генератора. У більшості МК AVR є вбудований внутрішній генератор, який калібрується, на основі ланцюжка резистори+конденсатори (RC-генератор) і для багатьох завдань його цілком вистачає. Наприклад при роботі ATmega8 можна не використовувати зовнішній кварц з конденсаторами. Крім того, для використання зовнішніх джерел тактового сигналу потрібно встановлювати правильні конфігураційні біти (fuse bits - внутрішньопрограмні перемикачі що задають початкові режими роботи МК).

3. Джерела живлення для МК

У разі використання програматора USB ASP живити МК можна прямо від нього. В даному випадку треба бути дуже уважним, оскільки невірно підключене або закорочене живлення від програматора може перевантажити USB порт ПК. До того ж USB порти різних версій мають обмеження по струму:

– USB 1.0 - 500 мА;

– USB 2.0 - 500 мА;

– USB 3.0 - 900 мА;

– USB 3.1 - до 5А.

Інтерфейс USB 1.0 є в старих комп'ютерах, USB 2.0 – в новіших комп’ютерах і ноутбуках, а USB 3.0 є в сучасних ноутбуках і материнській платі. Сили струму від USB порту цілком вистачить для живлення різних світлодіодів, цифрових табло, індикаторів, реле і навіть невеликих двигунів, але слід врахувати що можна зіпсувати порт у ноутбуці при його перевантаженні.

Блок живлення 5В/3.3В за допомогою штирків вставляється безпосередньо у макетну плату, рис. 4.

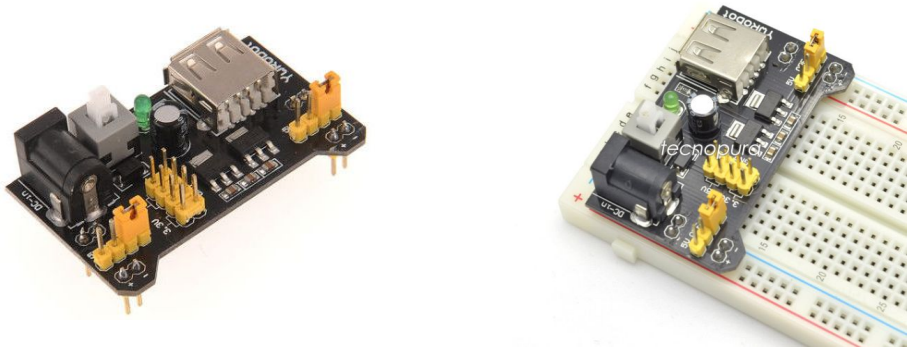


Рисунок 4 – Блок живлення 5В/3,3В для макетної плати

4. Мінімальне програмне забезпечення для програмування МК

Для програмування МК у середовищі **ОС Windows** потрібний безкоштовний програмний пакет **Atmel Studio** і ліцензійний програмний пакет **Proteus**.

Для програмування МК у середовищі **ОС Linux** потрібні наступні безкоштовні програми:

gcc-avr – це та частина потужної GNU Complier Collection (GCC), яка буде здійснювати крос-компіляцію для цільової архітектури мікроконтролерів AVR. GCC використовується для збирання самого Linux, він підтримує стандарти мови Сі ISO C90 (ANSI-C), ISO C99, а також власні розширення мови Сі. GCC підтримує величезну кількість архітектур процесорів. Код, скомпільований за допомогою GCC для однієї з платформ, потребує мінімальних правок (або не потребує правок взагалі) при компіляції для іншої цільової платформи.

binutils-avr – асемблер, компонувач і деякі корисні утиліти.

avr-libc – стандартна бібліотека Сі для Atmel AVR.

avrdude – утиліта для роботи з програматорами.

Це необхідний мінімум для створення застосунків для AVR і прошивки мікроконтролерів. Для налагодження програм використовуються наступні програми:

gdb-avr – налагоджувач для AVR.

avarice – інтерфейс між avr-gdb і AVR JTAG ICE.

simulavr – симулятор Atmel AVR.

Для встановлення потрібних програм у терміналі виконується наступна команда:

```
>sudo apt-get install gcc-avr binutils-avr avr-libc avrdude gdb-avr avarice simulavr
```

Перевірити результат встановлення пакетів можна командою:

```
>ls /usr/bin | grep avr
```

avr-addr2line, avr-ar, avr-as, avr-c++, avr-c++filt, avr-cpp, avrdude, avr-elfedit, avr-g++, avr-gcc, avr-gcc-4.9.2, avr-gcc-ar, avr-gcc-nm, avr-gcc-ranlib, avr-gcov, avr-gdb, avr-gprof, avr-ld, avr-ld.bfd, avr-man, avr-nm, avr-objcopy, avr-objdump, avr-ranlib, avr-readelf,avr-run, avr-size, avr-strings, avr-strip, simulavr, simulavr-disp, simulavr-vcd

Документацію по встановлених програмах можна отримати командою **man**<*ім'я програми*>:

```
>man avr-gcc
>man avr-objdump
>man avrdude.
```

Натиснувши клавішу 'h', при знаходженні всередині команди man, можна отримати список клавіш для переміщення по сторінці довідки.

Побачити розміщення пакетів можна командою **whereis:**

```
>whereis avrdude
avrdude: /usr/bin/avrdude /etc/avrdude.conf /usr/share/man/man1/avrdude.1.gz
```

5. Програматори

Для того щоб записати програму в МК AVR потрібний програматор. Найбільш поширеним способом програмування для МК AVR є внутрішньосхемне програмування (функція ISP – in-circuit serial programming) через комунікаційний порт SPI. Інтерфейс SPI має три лінії MOSI, MISO, SCK. Для програмування МК AVR використовуються різні програматори, наприклад STK500, STK600, USB ISP, USB ASP та інші.

Програматор – це невелика електронна схема, яка дозволяє підключити МК до одного з портів комп'ютера (COM, LPT, USB) для подальшого зчитування і записування прошивки (програмування). Найбільш надійно і зручно підключати програматор до USB-порту. Призначення виводів в USB ASP програматорі, рис. 5:

- VCC - плюс живлення, як правило +5В;
- GND - мінус живлення, земля (Ground);
- MOSI - вхід даних (Master Out Slave In);
-MISO - вихід даних (Master In Slave Out);
- SCK - тактовий сигнал (Serial Clock);
- RST - для подачі сигналу скидування (Reset);
- NC – не використовується.

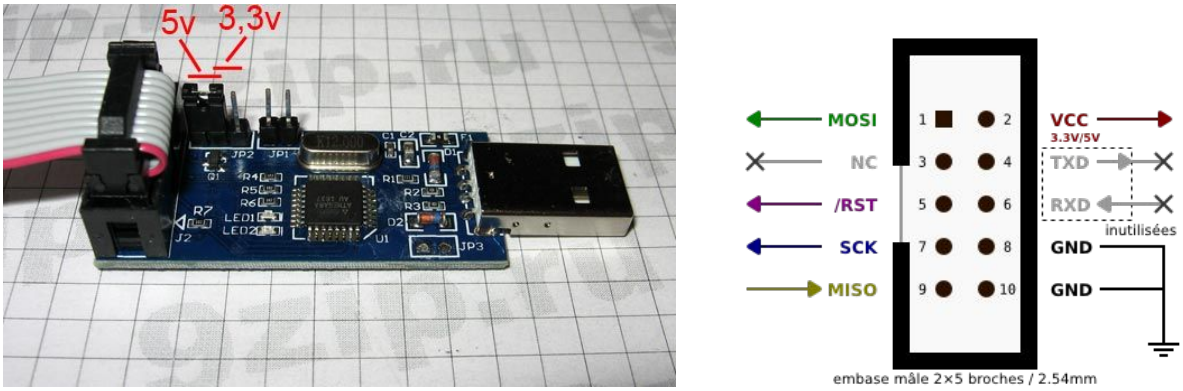


Рисунок 5 – Програматор USB ASP і його виводи

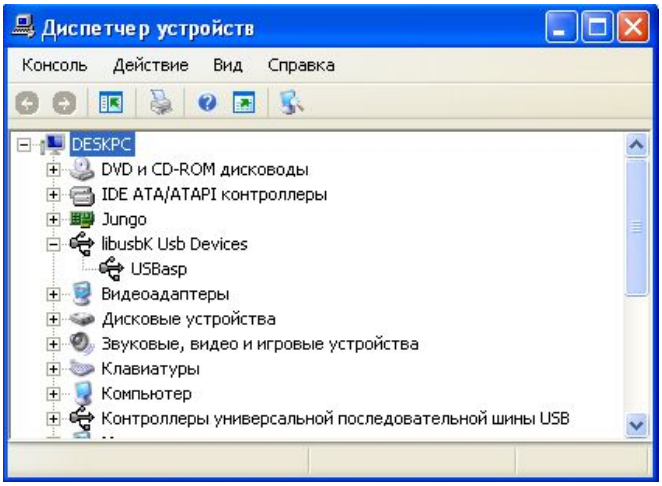
Призначення виводів в USB ISP програматорі, рис. 6:
- VCC – плюс живлення, як правило +5В;
- GND – мінус живлення, земля (Ground);
- MOSI – вхід даних (Master Out Slave In);
- MISO – вихід даних (Master In Slave Out);
- SCK – тактовий сигнал (Serial Clock);
- RST – для подачі сигналу скидування (Reset).

Рисунок 6 – USB ISP програматор із шлейфом для програмування

Даний програматор безпечний у використанні, має невеликі розміри і підтримується більшістю програм для прошивки МК AVR. USB ISP працює з ОС Linux, Mac OS X і Windows. Для Linux ніяких драйверів встановлювати не потрібно, після підключення програматора до USB порту пристрій зразу ж визначиться і буде готовий до використання. При програмуванні програматор виступає як ведучий пристрій, а мікроконтролер як ведений. Програматор USB ISP (китайська копія) працює тільки з оболонкою Prog ISP, і не підтримується на відміну від USB ASP програмою Atmel Studio.

6. Підключення програматора до ОС

Для підключення програматорів USB ASP, USB ISP до ОС Windows XP, 7, 10 потрібно розпакувати архів **usbasp-win-driver-x86-64-v3.07.zip**, вставити програматор в USB порт і інсталювати драйвер. В результаті інсталяції в диспетчері пристроїв появиться новий пристрій USBasp.



В ОС Linux драйвера не потрібно. Після під'єднання програматора до роз'єму USB, у файловій системі появляється новий пристрій. У Linux пристрої відображаються у файлову систему і тому можна читати, писати в них і змінювати права доступу так як робиться із звичайними файлами. Пристроїв у каталозі `/dev` достатньо багато , тому, щоб виявити нові, скористаємося стандартними консольними утилітами в терміналі:

Зберегти список пристроїв з `/dev` у тимчасовий файл `/tmp/dev1`

```
>ls /dev > /tmp/dev1
```

Приєднати програматор і зберегти список нових пристроїв з `/dev` у тимчасовий файл `/tmp/dev2`

```
>ls /dev > /tmp/dev2
```

Виявити різницю у списках пристроїв

```
>comm -3 /tmp/dev1 /tmp/dev2
```

У каталозі `/dev` появиться новий пристрій ttyUSB0. Переглянути дозволи на роботу з пристроєм можна командою:

```
>ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 окт.  22 14:49 /dev/ttyUSB0
```

Як видно, пристрою задано дозволи на читання і запис для користувача root і групи dialout. Користувачі, які не входять у цю групу, працювати з пристроєм не зможуть. Перевіримо список груп, в які входить поточний користувач:

```
>groups
owlet adm cdrom sudo dip plugdev lpadmin sambashare vboxusers
```

Так як поточний користувач не входить в групу `dialout`, то потрібно його у цю групу добавити:

```
>sudo usermod -a -G dialout `whoami`
```

Завершити поточний сеанс користувача і розпочати новий. Перевірити належність до групи:

```
>groups
owlet adm dialout cdrom sudo dip plugdev lpadmin sambashare vboxusers
```

Не у всіх дистрибутивах ОС Linux програматори USB ASP, USB ISP детектуються як пристрої ttyUSB0. У цьому випадку потрібно виконати додаткові налаштування системи.

Створити файл правил `99-USBasp.rules` у каталозі `/etc/udev/rules.d` і записати в нього наступні рядки:

```
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="16c0",
ATTRS{idProduct}=="05dc", MODE="0666", SYMLINK="USBasp"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="03eb",
ATTRS{idProduct}=="c8b4", MODE="0666", SYMLINK="USBisp"
```

Вивести вміст файлу для перевірки:

```
> cat /etc/udev/rules.d/99-USBasp.rules
# USBasp - USB programmer for Atmel AVR controllers
# Copy this file to /etc/udev/rules.d

SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="16c0",
ATTRS{idProduct}=="05dc", MODE="0666", SYMLINK="USBasp"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device",
ATTRS{idVendor}=="03eb",ATTRS{idProduct}=="c8b4", MODE="0666",
SYMLINK="USBisp"
```

Підключити програматор і виконати команду

```
>service systemd-udevd restart
```

Перевірити чи програматор появився серед USB пристроїв

```
>lsusb
...
Bus 004 Device 004: ID 16c0:05dc Van Ooijen Technische Informatica shared ID
for use with libusb
...
```

Перевірити символьне ім’я програматора, яке посилається на USB пристрій

```
>dmesg | grep USBasp
[    4.527467] usb 4-1: Product: USBasp
[   15.807043] usb 4-1: Product: USBasp
```

Вивести значення атрибутів пристрою

```
>udevadm info -a -n /dev/USBasp
looking at device '/devices/pci0000:00/0000:00:1d.0/usb4/4-1':
KERNEL=="4-1"
SUBSYSTEM=="usb"
DRIVER=="usb"
ATTR{authorized}=="1"
ATTR{avoid_reset_quirk}=="0"
ATTR{bConfigurationValue}=="1"
ATTR{bDeviceClass}=="ff"
ATTR{bDeviceProtocol}=="00"
ATTR{bDeviceSubClass}=="00"
ATTR{bMaxPacketSize0}=="8"
ATTR{bMaxPower}=="50mA"
ATTR{bNumConfigurations}=="1"
ATTR{bNumInterfaces}==" 1"
```

```
ATTR{bcdDevice}=="0102"
ATTR{bmAttributes}=="80"
ATTR{busnum}=="4"
ATTR{configuration}==""
ATTR{devnum}=="4"
ATTR{devpath}=="1"
ATTR{idProduct}=="05dc"
ATTR{idVendor}=="16c0"
ATTR{ltm_capable}=="no"
ATTR{manufacturer}=="www.fischl.de"
ATTR{maxchild}=="0"
ATTR{product}=="USBasp"
ATTR{quirks}=="0x0"
ATTR{removable}=="unknown"
ATTR{speed}=="1.5"
ATTR{urbnum}=="9"
ATTR{version}==" 1.10"
```

Порівняти права доступу і власників символьного посилання і USB пристрою

```
>ls -l /dev/USBasp
lrwxrwxrwx 1 root root 15 Sep  9 15:56 /dev/USBasp -> bus/usb/004/004

>ls -l /dev/bus/usb/004/004
crw-rw-rw-+ 1 root root 189, 387 Sep  9 15:56 /dev/bus/usb/004/004
```

У цьому випадку для роботи програматора пристрій `ttyUSB0` не потрібний.

7. З'єднання виводів мікросхеми МК і програматора

Щоб правильно з'єднати виводи мікросхеми МК і програматора потрібно вияснити маркування і призначення їх виводів. Вичерпну інформацію про МК можна знайти на офіційному сайті виробника `http://www.microchip.com` у документі який називається технічне описання (datasheet).

Для підключення МК АМК до програматора треба з'єднати виводи інтерфейсу ISP: VCC, GND, MISO, MOSI, SCK, RST, рис. 7.

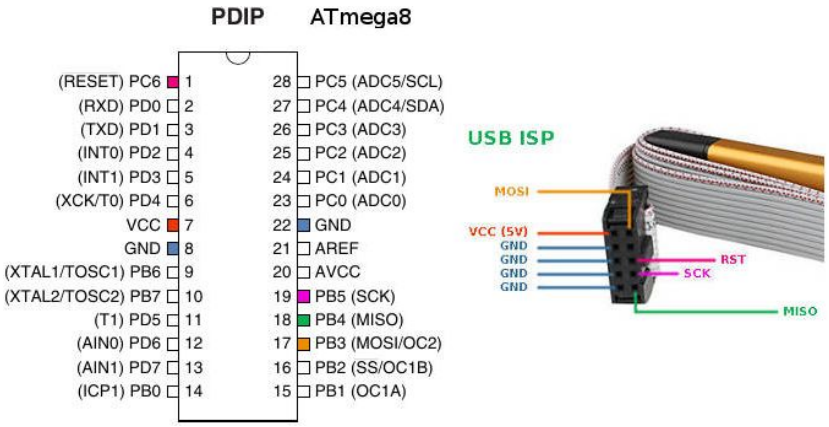


Рисунок 7 – Виводи МК ATmega8 і програматора (інтерфейс ISP).

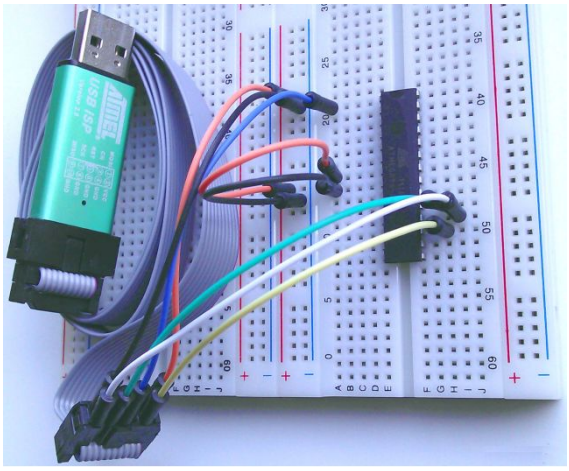


Рисунок 8 – Підключення програматора до МК ATmega8.

Показаного на рис.8 підключення достатньо для запису прошивки в МК. За замовчуванням МК ATmega8 використовує внутрішню RC-ланку для задання частоти тактового генератора, тому не потрібно встановлювати зовнішній кварц і конденсатори.

8. Програма AVRDUDE

Програма AVRDude(AVR Downloader - Uploader) – це дуже потужний кросплатформовий інструмент, який дозволяє програмувати усю лінійку МК Atmel, підтримуючи майже усі типи доступних зараз програматорів. Програма працює з консолі, що дозволяє автоматизувати процес прошивки МК, але вимагає при цьому уважності і навичок роботи з терміналом. Програма AVRDude запускається і працює з ОС Linux, Windows, MacOS, FreeBSD і іншими. Адреса офіційного сайту програми AVRDude : `http://www.nongnu.org/avrdude`.

8.1. Параметри запуску avrdude

Запустивши в консолі програму avrdude без аргументів можна побачити список доступних опцій для використання.

>avrdude

```
avrdude: option requires an argument -- 'l'
Usage: avrdude [options]
Options:
-p <partno>                Required. Specify AVR device.
-b <baudrate>              Override RS-232 baud rate.
-B <bitclock>              Specify JTAG/STK500v2 bit clock period (us).
-C <config-file>           Specify location of configuration file.
-c <programmer>            Specify programmer type.
-D                         Disable auto erase for flash memory
-i <delay>                 ISP Clock Delay [in microseconds]
-P <port>                  Specify connection port.
-F                         Override invalid signature check.
-e                         Perform a chip erase.
-O                         Perform RC oscillator calibration (see AVR053).
-U <memtype>:r|w|v:<filename>[:format]
Memory operation specification.
```

```
Multiple -U options are allowed, each request
is performed in the order specified.
-n                          Do not write anything to the device.
-V                          Do not verify.
-u                          Disable safemode, default when running from a script.
-s                          Silent safemode operation, will not ask you if
fuses should be changed back.
-t                          Enter terminal mode.
-E <exitspec>[,<exitspec>] List programmer exit specifications.
-x <extended_param>         Pass <extended_param> to programmer.
-y                          Count # erase cycles in EEPROM.
-Y <number>                 Initialize erase cycle # in EEPROM.
-v                          Verbose output. -v -v for more.
-q                          Quell progress output. -q -q for less.
-l logfile                  Use logfile rather than stderr for diagnostics.
-?                          Display this usage.

avrdude version 6.2, URL: <http://savannah.nongnu.org/projects/avrdude/>
```

Розглянемо всі опції програми:

- **-p <partno>** – обов'язкова опція, де <partno> вказує короткий псевдонім МК
- **-b <baudrate>** – дозволяє перевизначити вказану для програматора в конфігурації програми швидкість підключення по інтерфейсу RS-232;
- **-B <bitclock>** – задає Bit Clock Period для інтерфейсу налагодження JTAG або ISP Clock (тільки для JTAG ICE). Значення <bitclock> вказується в мікросекундах, для JTAG ICE за замовчуванням встановлено 1 мікросекунда і підходить для МК працюючих на частотах 4МГц і вище;
- **-C <config-file>** – як <config-file> вказується повний шлях до файлу конфігурації з необхідними налаштуваннями програми. За замовчуванням використовується файл /etc/avrdude.conf (Linux);
- **-c <programmer>** – як <programmer> вказується псевдонім використовуваного програматора, наприклад "usbasp".
- **-D** - опція забороняє автоматичне стирання Flash-пам'яті. Автоматичне стирання не використовується в МК родини ATxmega;
- **-i <delay>** – встановлення паузи перед кожним відправленням кожного біту для bitbang-програматорів. Як <delay> вказується значення в мікросекундах. Це буває необхідно, якщо для програмування використовується комп'ютер з дуже швидким процесором або ж МК с низькою тактовою частотою (32КГц, 128КГц), що дозволяє витримувати умову: частота ISP < 1/4 частоти процесора;
- **-P <port>** – як значення <port> вказується використовуваний програматором порт. За замовчуванням використовується /dev/ppi0 (паралельний порт) і /dev/cuaa0 (послідовний порт) в залежності від програматора;
- **-F** – опція для відключення перевірки сигнатури МК. За замовчуванням перед програмуванням avrdude перевіряє сигнатуру МК, яка в деяких випадках може бути пошкоджена, але при цьому МК може продовжувати нормально функціонувати;
- **-e** – стирання вмісту FLASH і EEPROM пам'яті (заповнення значеннями 0xFF), очищення конфігураційних бітів (fuse-bits). Винятком є МК родини ATxmega в яких використовується посторінковий запис;

- **-O** – опція для калібрування RC-генератора у відповідності з зауваженнями AVR053 від Atmel. Підтримується тільки на програматорах STK500v2, AVRISP mkII, і JTAG ICE mkII. Результат буде збережений в EEPROM пам'яті в комірці з адресою 0;
- **-U <memtype>:r|w|v:<filename>[:format]** - комплексна опція для задання виконуваної операції з пам'яттю (читання, записування, перевірка);
- **-n** – заборона на записування в МК, використовується для налагодження avrdude;
- **-V** – відключення автоматичної перевірки записаної інформації;
- **-u** – відключення режиму безпечної (safe mode) перевірки і співставлення комірок конфігурації (fuse bits) до і після завершення програмування. Дана опція необхідна якщо потрібно змінити значення конфігураційних бітів (fuse bits), в іншому випадку avrdude як міру безпеки змінить їх значення на ті, які були прочитані перед програмуванням;
- **-s** - заборона виведення запитів в безпечному режимі при роботі з конфігураційними бітами;
- **-t** – переводить avrdude в режим терміналу (terminal mode);
- **-E <exitspec>[,<exitspec>]** – зміна стану ліній паралельного порту після програмування на вказані в аргументах опції. За замовчуванням встановлюється ті стани ліній, які були до початку роботи;
- **-x <extended_param>** – дозволяє вказати додатковий спеціальний параметр для використовуваного програматора;
- **-y** – опція, яка включає збереження кількості стирань МК в останні 4 байти пам'яті EEPROM;
- **-Y <number>** – вказане значення <number> буде збережене як число циклів-стирань МК в пам'яті EEPROM;
- **-v** – розширене виведення інформації про роботу програми (verbose);
- **-q** – відключає відображення смужки прогресу при операціях з МК. Для ще більшого зменшення відображуваної інформації опцію потрібно вказати двічі;
- **-l <logfile>** – перенаправлення всього виведення для налагодження у вказаний файл, де <logfile> – повний шлях до файлу для збереження даних;
- **?** – відображення довідки.

8.2. Моделі МК

Значення параметру	Модель МК
uc3a0512	AT32UC3A0512
c128	AT90CAN128
c32	AT90CAN32
c64	AT90CAN64
pwm2	AT90PWM2
pwm2b	AT90PWM2B
pwm3	AT90PWM3
pwm316	AT90PWM316
pwm3b	AT90PWM3B

1200	AT90S1200
2313	AT90S2313
2333	AT90S2333
2343	AT90S2343
4414	AT90S4414
4433	AT90S4433
4434	AT90S4434
8515	AT90S8515
8535	AT90S8535
usb1286	AT90USB1286
usb1287	AT90USB1287
usb162	AT90USB162
usb646	AT90USB646
usb647	AT90USB647
usb82	AT90USB82
m103	ATmega103
m128	ATmega128
m1280	ATmega1280
m1281	ATmega1281
m1284p	ATmega1284P
m1284rfr2	ATmega1284RFR2
m128rfa1	ATmega128RFA1
m128rfr2	ATmega128RFR2
m16	ATmega16
m161	ATmega161
m162	ATmega162
m163	ATmega163
m164p	ATmega164P
m168	ATmega168
m168p	ATmega168P
m169	ATmega169
m16u2	ATmega16U2
m2560	ATmega2560
m2561	ATmega2561
m2564rfr2	ATmega2564RFR2
m256rfr2	ATmega256RFR2
m32	ATmega32
m324p	ATmega324P

m324pa	ATmega324PA
m325	ATmega325
m3250	ATmega3250
m328	ATmega328
m328p	ATmega328P
m329	ATmega329
m3290	ATmega3290
m3290p	ATmega3290P
m329p	ATmega329P
m32u2	ATmega32U2
m32u4	ATmega32U4
m406	ATMEGA406
m48	ATmega48
m48p	ATmega48P
m64	ATmega64
m640	ATmega640
m644	ATmega644
m644p	ATmega644P
m644rfr2	ATmega644RFR2
m645	ATmega645
m6450	ATmega6450
m649	ATmega649
m6490	ATmega6490
m64rfr2	ATmega64RFR2
m8	ATmega8
m8515	ATmega8515
m8535	ATmega8535
m88	ATmega88
m88p	ATmega88P
m8u2	ATmega8U2
t10	ATtiny10
t11	ATtiny11
t12	ATtiny12
t13	ATtiny13
t15	ATtiny15
t1634	ATtiny1634
t20	ATtiny20
t2313	ATtiny2313

t24	ATtiny24
t25	ATtiny25
t26	ATtiny26
t261	ATtiny261
t4	ATtiny4
t40	ATtiny40
t4313	ATtiny4313
t43u	ATtiny43u
t44	ATtiny44
t45	ATtiny45
t461	ATtiny461
t5	ATtiny5
t84	ATtiny84
t85	ATtiny85
t861	ATtiny861
t88	ATtiny88
t9	ATtiny9
x128a1	ATxmega128A1
x128a1d	ATxmega128A1revD
x128a1u	ATxmega128A1U
x128a3	ATxmega128A3
x128a3u	ATxmega128A3U
x128a4	ATxmega128A4
x128a4u	ATxmega128A4U
x128b1	ATxmega128B1
x128b3	ATxmega128B3
x128c3	ATxmega128C3
x128d3	ATxmega128D3
x128d4	ATxmega128D4
x16a4	ATxmega16A4
x16a4u	ATxmega16A4U
x16c4	ATxmega16C4
x16d4	ATxmega16D4
x16e5	ATxmega16E5
x192a1	ATxmega192A1
x192a3	ATxmega192A3
x192a3u	ATxmega192A3U
x192c3	ATxmega192C3

x192d3	ATxmega192D3
x256a1	ATxmega256A1
x256a3	ATxmega256A3
x256a3b	ATxmega256A3B
x256a3bu	ATxmega256A3BU
x256a3u	ATxmega256A3U
x256c3	ATxmega256C3
x256d3	ATxmega256D3
x32a4	ATxmega32A4
x32a4u	ATxmega32A4U
x32c4	ATxmega32C4
x32d4	ATxmega32D4
x32e5	ATxmega32E5
x384c3	ATxmega384C3
x384d3	ATxmega384D3
x64a1	ATxmega64A1
x64a1u	ATxmega64A1U
x64a3	ATxmega64A3
x64a3u	ATxmega64A3U
x64a4	ATxmega64A4
x64a4u	ATxmega64A4U
x64b1	ATxmega64B1
x64b3	ATxmega64B3
x64c3	ATxmega64C3
x64d3	ATxmega64D3
x64d4	ATxmega64D4
x8e5	ATxmega8E5

8.3. Типи програматорів (опція `-c <programmer>`)

Налаштування всіх програматорів міститься у конфігураційному файлі за замовчуванням. Переглянуто його вміст в ОС Linux можна командою:

```
>less /etc/avrdude.conf
...
programmer
id    = "usbasp";
desc  = "USBasp, http://www.fischl.de/usbasp/";
type  = "usbasp";
connection_type = usb;
usbvid     = 0x16C0; # VOTI
usbpid     = 0x05DC; # Obdev's free shared PID
```

```
usbvendor  = "www.fischl.de";
usbproduct = "USBasp";

# following variants are autodetected for id "usbasp"

# original usbasp from fischl.de
# see above "usbasp"

# old usbasp from fischl.de
#usbvid     = 0x03EB; # ATMEL
#usbpid     = 0xC7B4; # (unoffical) USBasp
#usbvendor  = "www.fischl.de";
#usbproduct = "USBasp";

# NIBObee (only if -P nibobee is given on command line)
# see below "nibobee"
;
...
```

Нижче показані значення опції і відповідні дані програматорів:

Опція -c	Програматор
abcmini	ABCmini Board, aka Dick Smith HOTCHIP
alf	Nightshade ALF-PgmAVR, http://nightshade.homeip.net/
arduino	Плата Arduino, протокол подібний з STK500 1.x
atisp	AT-ISP V1.1 кабель програмування для AVR-SDK1, http://micro-research.co.th/
avr109	Atmel AppNote AVR109 Boot Loader
avr910	Atmel Low Cost Serial Programmer
avr911	Atmel AppNote AVR911 AVROSP (an alias for avr109)
avrftdi	FTDI MPSSE (FT2232 etc.) підтримка bitbang
avrisp	Atmel AVR ISP (псевдонім для stk500)
avrisp2	Atmel AVR ISP mkII (псевдонім для stk500v2)
avrispmkII	Atmel AVR ISP mkII (псевдонім для stk500v2)
avrispv2	Atmel AVR ISP, running a version 2.x firmware (an alias for stk500v2)
bascom	Bascom SAMPLE programming cable
blaster	Altera ByteBlaster
bsd	Brian Dean's Programmer, http://www.bsdhome.com/avrdude/
butterfly	Atmel Butterfly Development Board
c2n232i	C2N232I, reset=dtr sck=!rts mosi=!txd miso=!cts,
dapa	Direct AVR Parallel Access cable
dasa	serial port banging, reset=rts sck=dtr mosi=txd miso=cts
dasa3	serial port banging, reset=!dtr sck=rts mosi=txd miso=cts
dragon_dw	AVR Dragon in debugWire mode
dragon_hvsp	AVR Dragon in high-voltage serial programming mode
dragon_isp	AVR Dragon в режимі ISP

dragon_jtag	AVR Dragon в режимі JTAG
dragon_pp	AVR Dragon in (high-voltage) parallel programming mode
dt006	Dontronics DT006
ere-isp-avr	ERE ISP-AVR
frank-stk200	Клон STK200 от Frank'a, http://electropol.free.fr/spip/spip.php?article15
futurlec	Кабель програмування Futurlec.com
jtag1	Atmel JTAG ICE mkI, running at 115200 Bd
jtag1slow	Atmel JTAG ICE mkI, running at 19200 Bd
jtag2slow	Atmel JTAG ICE mkII (default speed 19200 Bd)
jtag2,jtag2fast	Atmel JTAG ICE mkII, running at 115200 Bd
jtag2isp	Atmel JTAG ICE mkII in ISP mode.
jtag2dw	Atmel JTAG ICE mkII in debugWire mode.
jtagmkI	Atmel JTAG ICE mkI, running at 115200 Bd
jtagmkII	Atmel JTAG ICE mkII (default speed 19200 Bd)
mib510	Crossbow MIB510 programming board
pavr	Jason Kyle's pAVR Serial Programmer
picoweb	Picoweb Programming Cable, http://www.picoweb.net/
pony-stk200	Pony Prog STK200
ponyser	design ponyprog serial, reset=!txd sck=rts mosi=dtr miso=cts
siprog	Lancos SI-Prog, http://www.lancos.com/siprogsch.html
sp12	Steve Bolt's Programmer
stk200	STK200
stk500	Atmel STK500, probing for either version 1.x or 2.x firmware
stk500hvsp	Atmel STK500 в режимі високовольтного послідовного програмування (high-voltage serial programming mode), тільки для прошивок версії 2.x
stk500pp	Atmel STK500 в режимі паралельного програмування (parallel programming) , тільки прошивка версії 2.x
stk500v1	Atmel STK500, з версією прошивки 1.x
stk500v2	Atmel STK500, з версією прошивки 2.x
stk600	Atmel STK600 в режимі ISP або в PDI режимі для пристрою ATxmega
stk600hvsp	Atmel STK600 в режимі високовольтного послідовного програмування (high-voltage serial programming mode)
stk600pp	Atmel STK600 в режимі паралельного програмування (parallel programming)
usbasp	USBasp, http://www.fischl.de/usbasp/
usbtiny	USBtiny - простий USB програматор, http://www.ladyada.net/make/usbtinyisp/
xil	Xilinx JTAG кабель

8.3. Робота з пам’яттю (опція `-U <memtype>:r|w|v:<filename>[:format]`)

Як <memtype> вказується тип пам’яті для роботи:

- calibration – байти калібрування RC-генератора (один або декілька);
- eeprom – енергонезалежна пам’ять (EEPROM) МК;
- efuse – додатковий конфігураційний біт;
- flash – FLASH пам’ять МК;
- fuse – конфігураційний байт МК тільки с одним fuse-байтом;
- hfuse – старший fuse-байт;
- lfuse – молодший fuse-байт;
- lock – байт блокування;
- signature – три байти які позначають сигнатуру кристалу (device ID);
- fuseN – байт з конфігураційними бітами для ATxmega мікросхем, N - ціле число для кожного ф’юза який підтримується пристроєм;
- application – область застосунку у Flash пам’яті для МК ATxmega;
- apptable – таблиця застосунків в області Flash пам’яті для пристроїв ATxmega;
- boot – завантажувальна область Flash пам’яті для пристроїв ATxmega;
- prodsig – область з заводською сигнатурою (calibration) для пристроїв ATxmega;
- usersig – область користувацької сигнатури для пристроїв ATxmega.

Далі через двокрапку задається виконувана операція з пам’яттю МК:

1. **r** – прочитати вказану область пам’яті і записати у вказаний файл <filename>;
2. **w** – прочитати дані з файлу <filename> і записати у вказану пам’ять пристрою;
3. **v** – прочитати дані з вказаного файлу <filename> і з вказаної області пам’яті (verify, перевірка).

В полі <filename> вказується повний або відносний шлях до файлу, який використовується для записування або читання даних. Поле ":format" є не обов’язковим, і вказує формат використовуваного файлу:

- i – Intel HEX;
- s – Motorola S-record;
- r – raw binary (RAW формат);
- e – ELF (Executable and Linkable Format);
- m – значення байтів для записування вказуються безпосередньо у командному рядку в полі <filename> і розділяються пропусками або комами. За замовчуванням байти пишуться у десятковій системі, якщо вказати 0x - будуть записані шістнадцяткові значення, а якщо перед байтом стоїть 0 - буде записане вісімкове число;
- a – авто-визначення формату (auto detect);
- d – десятковий формат (decimal), числа відокремлюються комами;
- h – шістнадцятковий формат (hexadecimal), числа починаються з 0x;
- o – вісімковий формат (octal), перед числами ставиться 0;
- b – двійковий формат (binary), перед числами ставиться 0b.

За замовчуванням використовується автоматичне визначення формату (auto detect).

8.4. Стан ліній паралельного порту (`-E <exitspec>[,<exitspec]`)

- **reset** – на лінії RESET буде низький рівень, МК залишиться у стані скидування;

- **noreset** – на лінію RESET поступить високий рівень для запуску МК після програмування;
- **vcc** – встановлення високого рівня на лінії порту VCC, який може використовуватися для живлення МК;
- **novcc** – подача низького рівня на лінію VCC.

Допускається використання декількох значень через кому.

8.5. Приклади використання avrdude

Протестувати зв'язок МК ATtiny13 з програматором USBASP:

```
>avrdude -p t13 -c usbasp
```

Прочитати Flash-пам'ять МК ATmega88 в нульовий пристрій (/dev/null), тест читання флеш-пам'яті:

```
>avrdude -p m88 -c usbasp -U flash:r:/dev/null:i
```

Прочитати Flash-пам'ять МК ATmega8 **у файл** формату Intel HEX - /tmp/flash_dump.hex, при цьому вказується, що для програматора потрібно використати саме USB-порт (-P usb) і виводити більше налагоджувальної інформації (-v):

```
>avrdude -p m8 -c usbasp -P usb -v -U flash:r:/tmp/flash_dump.hex:i
```

Прочитати вміст EEPROM-пам'яті МК ATtiny85 і зберегти його у файл RAW формату (/tmp/eeprom_dump.raw), використовуючи при цьому програматор USBTiny:

```
>avrdude -p t85 -c usbtiny -P usb -v -U eeprom:r:/tmp/eeprom_dump.raw:r
```

Записати дані з HEX-файлу (/tmp/program_m8.hex) **у FLASH-пам'ять** МК ATmega8, використовуючи програматор STK-500:

```
>avrdude -c stk500 -p m8 -U flash:w:/tmp/program_m8.hex
```

Записати дані у **FLASH і EEPROM** пам'ять однією командою, використовуючи як джерела даних файли `/tmp/flash_1.hex` і `/tmp/eeprom_1.hex`:

```
>avrdude -c stk500 -p m8 -U flash:w:/tmp/flash_1.hex -U
eeprom:w:/tmp/eeprom_1.hex
```

Прочитати ф'юзи з МК ATmega8 і зберегти їх у файли в шістнадцятковому форматі:

```
>avrdude -c usbasp -p m8 -U hfuse:r:m8_hfuse.txt:h -U lfuse:r:m8_lfuse.txt:h
```

Записати ф'юзи для МК ATmeag32 для заданої частоти внутрішнього RC-генератора 4 MHz (Low=0xc3, High=0x99):

```
>avrdude -c usbasp -p m32 -U lfuse:w:0xc3:m -U hfuse:w:0x99:m
```

Записати значення **бітів блокування** (Lock Bits) для МК ATTiny13, підключеного до програматора USBASP, встановити значення байта у 0xFC (11111100):

```
>avrdude -c usbasp -p t13 -U lock:w:0xFC:m
```

8.6. Графічна оболонка для avrdude

Для avrdude існує графічна оболонка AVR8 Burn-O-Mat. Вона дозволяє читати і записувати дані у Flash і EEPROM пам'ять, а також за її допомогою можна зручно і наглядно розраховувати біти ф'юзів і потім прошивати їх в МК. Програма написана на мові Java і працює в ОС Windows і GNU Linux. Офіційний сайт програми http://avr8-burn-o-mat.brischalle.de.

Для розрахунку ф'юзів є онлайн-калькулятор **http://www.engbedded.com/fusecalc**, а для розрахунку Lock-бітів **http://eleccelerator.com/fusecalc**.

9. Робота з конфігураційними (Fuse) і блокувальними (Lock) бітами в МК AVR

9.1. Аналоги Fuse і Lock бітів

Слово "Fuse" (ф'юз) с англійської перекладається як "запобіжник", а слово "lock" - блокування. Fuse і Lock біти в МК AVR можна уявити собі як внутрішньо мікросхемні перемикачі, які в одному стані контактів дають 1 (пропускають струм), а в іншому – 0 (не пропускають струм).

Рисунок 9 – Приклад запобіжників і їх кодування

Важливо пам'ятати що для МК AVR всі ф'юзи і біти блокування мають наступні значення:

- 1 – не встановлений (не запрограмований);
- 0 – встановлений (запрограмований).

9.2. Призначення бітів конфігурації і блокування

В МК AVR Fuse і Lock біти розміщуються в спеціальній області пам'яті. За допомогою Fuse-бітів можна встановити різні режими роботи МК, його виводів, задати джерело тактового сигналу і його параметри, перетворити вивід "Reset" у звичайний порт введення/виведення та багато чого іншого.

Деякі назви і описання часто використовуваних Fuse-бітів:

- **RSTDSBL** (ReSeT DiSaBLe) – "заборонити reset", перетворивши вивід скидування МК в звичайний порт введення/виведення;
- **CKSEL0..3** (ClocK SELect) – чотири біти для встановлення параметрів джерела тактового сигналу МК (зовнішній кварц, внутрішній RC-генератор, подільник частоти і т. п.);
- **CKDIV8** (ClocK DIVision 8) – якщо цей біт буде встановлений то тактова частота від внутрішнього RC-генератора буде ділитися на 8;
- **CKOPT** (ClocK OPTimization) – задає розмах сигналу (амплітуду) з вихідного тактового генератора, оптимізація споживаного струму впливає на завадостійкість;
- **SUT0..1** (Start Up Time) – встановлення часової затримки запуску програми після подачі живлення або перезапуску МК;
- **SPIEN** (Serial Programming Interface ENable) – дозвіл/заборона програмування МК через послідовний програмний інтерфейс;
- **JTAGEN** (JTAG ENable) – дозвіл/заборона використання JTAG інтерфейсу;
- **EESAVE** (EEprom SAVE) – якщо цей біт встановлений, то вміст енергонезалежної пам'яті буде збережено після стирання кристалу (опція -e в AVRDude);
- **WDTON** (Watch Dog Timer ON) – відключення програмного керування сторожовим таймером, таймер автоматично запускається при подачі живлення на МК;

- **BODEN** (Brown-Out Detection ENabled), **BODLEVEL** (Brown-Out Detection LEVEL) - біти для включення і налаштування моніторингу за напругою живлення МК;
- **BOOTRST** (BOOT ReSeT) - виконувати запуск через завантажувач (Boot Loader), МК почне виконувати програми не з адреси 0x0000 (за замовчуванням), а з адреси де розміщений завантажувач.

Важливо зауважити, що при встановленні ф'юза RSTDSBL втрачається можливість перепрошивки МК через ISP-інтерфейс. Тем не менш, з використанням високовольтного (+12В) паралельного програматора перепрошивка все ж можлива.

Біти блокування дозволяють встановити режими доступу (записування/читання) до внутрішньої Flash-пам'яті і/або EEPROM, причому напрямок доступу можна обмежити як зсередини МК, так і зовні (при використанні ISP-інтерфейсу). Дана можливість може бути корисна для захисту від копіювання/змін програми, а також даних, які зберігаються в EEPROM пам'яті.

9.3. Структура конфігураційного і блокувального байту

Конфігураційні Fuse-біти розміщуються в трьох байтах:

1. Fuse Low Byte – молодший байт;
2. Fuse High Byte – старший байт;
3. Fuse Extended Byte – байт с опціями розширених функцій.

Блокувальні Lock-біти розміщені в окремому блокувальному байті.

У кожної моделі МК свій набір доступних для зміни конфігураційних і блокувальних бітів, який описаний в офіційній документації (datasheet).

Приклад структури Fuse-байтів для МК ATTiny13**.** Перший рядок містить номер біту, другий – назву, а третій – значення за замовчуванням (значення 0 вказує, що біт запрограмований).

Молодший Fuse-байт (Fuse Low Byte)							
7	6	5	4	3	2	1	0
SPIEN	EESAVE	WDTON	CKDIV8	SUT1	SUT0	CKSEL1	CKSEL0
0	1	1	0	1	0	1	0

Як видно, інтерфейс послідовного програмування (SPI) за замовчуванням дозволений, частота внутрішнього тактового генератора ділиться на 8 (CKDIV8), як джерело тактового сигналу використовується внутрішній RC-генератор (CKSEL0).

Старший Fuse-байт (Fuse High Byte)							
7	6	5	4	3	2	1	0
-	-	-	SELFPRGEN	DWEN	BODLEVEL1	BODLEVEL0	SRTDISBL
1	1	1	1	1	1	1	1

У даному байті ф'юзів всі біти неактивні (не запрограмовані).

Структура блокувального байту (Lock Byte):

Старший Fuse-байт (Fuse High Byte)							
7	6	5	4	3	2	1	0

-	-	-	-	-	-	LB2	LB1
1	1	1	1	1	1	1	1

Встановлення біта 0 (LB1) в 0 заборонить програмування внутрішньої Flash і EEPROM пам’яті. Якщо ще додатково встановити біт 1 (LB2) то це додатково заблокує можливість зчитування даних з пам’яті МК ATTiny13.

9.4. Робота з конфігураційними Fuse- і Lock-бітами

Встановлювати конфігураційні Fuse- і блокувальні Lock-біти потрібно дуже акуратно і з розумінням для чого це робиться. Некоректне встановлення бітів може вивести з ладу МК!

Рекомендована послідовність записування конфігураційних і блокувальних бітів:

- прочитати значення потрібного байта з МК;
- встановити в отриманому байті потрібні біти;
- записати результуючий байт в МК.

Чому використовується саме така послідовність?. Тому, що при встановленні окремого біту можна порушити стан інших бітів, наприклад перемикання МК на джерело тактового сигналу, який не передбачений і МК просто не запуститься.

Важливо пам’ятати, що у випадку виконання операції стирання (опція "-e" в AVRDUDE) всі Fuse- і Lock-біти будуть відновлені до стану за замовчуванням, а записана у Flash-пам’ять програма витерта.

Приклад встановлення Fuse-бітів з використанням програми AVRDude. Наприклад, потрібно скинути конфігураційний біт CKDIV8 в молодшому байті щоб МК ATTiny13 почав працювати на частоті 8 МГц замість 1 МГц. Для цього підключається МК до програматора, рис. 11, і виконується команда

```
>avrdude -p t13 -c usbasp
```

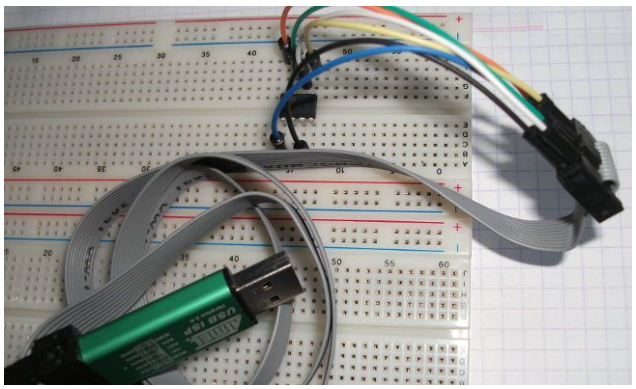


Рисунок 10 – Підключення МК ATTiny13 до програматора USB ISP

За замовчуванням при запуску AVRDude виводить на екран значення всіх трьох конфігураційних байтів.

```
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s
```

```
avrdude: Device signature = 0x1e9007
avrdude: safemode: Fuses OK (E:FF, H:FF, L:6A)
avrdude done.  Thank you.
None
```

Як видно, значення розширеного Fuse-байта (E) - FF (0xFF), старшого (H) - FF, молодшого (L) - 6A (0x6A).

Значення любого з конфігураційних байтів можна зчитати і записати у файл. Наприклад, запишемо значення молодшого конфігураційного байта у файл "lfuse.txt" і виведемо його вміст на екран:

```
>avrdude -p t13 -c usbasp -U lfuse:r:lfuse.txt:h
>cat lfuse.txt
```

Значення молодшого конфігураційного байта - 0x6A, що у двійковому поданні дорівнює 01101010 (значення за замовчуванням).

Тепер потрібно змінити значення біта CKDIV8 в байті 011**0**1010 на 1 (скинути його), отримуємо двійкове число 011**1**1010, яке дорівнює 0x7A. Записуємо нове значення байта в МК:

```
>avrdude -p t13 -c usbasp -U lfuse:w:0x7a:m
```

Значення параметра `"-U" - "lfuse:w:0x7a:m"` означає що виконується операція з пам'яттю (-U), а саме з молодшим байтом, який містить біти (lfuse), будемо записувати (:w) значення "0x7a", вказавши його прямо в командному рядку (:m). Результат виконання команди:

```
avrdude: warning: cannot set sck period. please check for usbasp firmware
update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9007
avrdude: reading input file "0x7a"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0x7a:
avrdude: load data lfuse data from input file 0x7a:
avrdude: input file 0x7a contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified

avrdude: safemode: Fuses OK (E:FF, H:FF, L:7A)
avrdude done.  Thank you.
None
```

Як видно, записано і прочитано – "Fuses OK (E:FF, H:FF, L:**7A**)"! Тепер МК буде працювати на частоті 8МГц.

Приклад встановлення Lock Bits з використанням AVRDude. Для встановлення блокувальних бітів в МК використовується та ж послідовність. Зчитується поточне значення байта з блокувальними бітами у файл lock.txt:

```
>avrdude -p t13 -c usbasp -U lock:r:lock.txt:h
>cat lock.txt

avrdude: warning: cannot set sck period. please check for usbasp firmware
update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9007
avrdude: reading lock memory:

Reading | ################################################## | 100% 0.00s

avrdude: writing output file "lock.txt"
avrdude: safemode: Fuses OK (E:FF, H:FF, L:7A)
avrdude done.  Thank you.

0x3f
None
```

Поточне значення байта з Lock Bits - 0x3F у двійковому поданні дорівнює 00111111. В технічній документації (datasheet) ATTiny13 вказано, що для змін дієві тільки два біти - 00111**11**. Встановлення цих бітів в 0 заблокує можливість програмування і читання внутрішньої пам'яті Flash і EEPROM. В результаті отримаємо значення бітів 00111**00**, а в шістнадцятковому поданні - 0x3C.

Для запису Lock-байту потрібно виконати наступну команду:

```
>avrdude -p t13 -c usbasp -U lock:w:0x3c:m
```

Потрібно бути гранично уважним при роботі з цією командою, оскільки некоректні зміни Lock-бітів можуть мати незворотні наслідки!

9.5. Ручний розрахунок бітів у конфігураційному байті

Припустимо, що для МК ATTiny2313 потрібно виконати наступне:

- Скинути біт 4 (CKDIV8), тобто встановити в 1;
- Активувати біт 6 (EESAVE), тобто встановити в 0.

Таблиця 1 – Відповідність двійкових і шістнадцяткових подань чисел

Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Для нового МК значення молодшого конфігураційного Fuse-байта за замовчуванням - 0x6a.

Його потрібно перевести у двійкове подання. Розділюємо значення "6A" (0x6a) на дві частини – "6" і "A", знаходимо у табл. 1 відповідні їм двійкові подання – 0110 (6) і 1010 (A). Значення 0x6a в двійковій системі числення – 01101010.

Встановлюємо 4-й біт в 1, а 6-й біт в 0: [7й біт=>] 0**0**1**1**1010 [<=0й біт]. Розділюємо отримане двійкове подання 00111010 на дві рівні частини: 0011 і 1010. Перетворюємо ці частини в шістнадцяткове подання використовуючи табл. 1: 0011 = 3, 1010 = A. З'єднуємо отримані цифри в шістнадцятковий формат: 0011 1010 = 3A. Отже, для того щоб скинути біт 4 (CKDIV8) в 1 і активувати біт 6 (EESAVE) в 0 потрібно записати у молодший конфігураційний байт значення "0x3a".

Завдання.

1. Встановити програмне забезпечення в Linux або Windows для роботи з МК AVR.
2. Вивчити послідовності створення проектів, компіляції програм та налаштування програм.
3. Вивчити послідовність запису програми в МК.
4. Вивчити налаштування Fuse-бітів і блокувальних бітів.

Питання.

1. Архітектура і основні структурні елементи МК AVR.
2. Функції основних структурних елементів МК AVR.
3. Мінімальне апаратне забезпечення для вивчення МК AVR.
4. Мінімальне програмне забезпечення для програмування МК AtVR.
5. Програматори і підключення їх до ОС.
6. Програма AVER Dude і її опції.
7. Призначення конфігураційних і блокувальних бітів в МК AVR.
8. Ручний розрахунок конфігураційних бітів в МК AVR.

ЛАБОРАТОРНА РОБОТА № 2. Асемблер, система команд, адресація даних

Мета роботи: вивчення системи команд і адресації даних МК ATmegs8/ATmega48/ATmega8515/ATmega8535, інтегрованого середовища розроблення програм

1. Теоретична частина

1.1. Асемблер

Асемблер транслює код програми користувача (*.asm) в об'єктний (*.obj),), який потім компонується у бінарний виконуваний код (*.hex). Виконуваний код можна протестувати в середовищі AVR Studio і після цього завантажити (за допомогою інструментального набору STK500, USBasp, USBisp) у флеш пам'ять або EEPROM пам'ять МК. AVR асемблер генерує виконуваний код з абсолютною адресацією, тому непотрібний крок компонування. AVR асемблер не розпізнає регістр букв.

Кожна команда асемблера переважно записується в один рядок довжиною до 120 знаків. Рядки можуть містити наступні елементи:

• Позначки (точки для переходів) – алфавітно-цифрові знаки, які мають починатися з букви і закінчуватися двокрапкою.

• Мнемоніки команд – символічного подання машинних команд.

• Директиви (вказівки для асемблера) – визначена послідовність символів, яка починається з крапки і не транслюється в машинні команди, а тільки використовується в процесі асемблювання.

• Коментарі – починаються з символу ";" і використовуються в програмі для кращого її розуміння.

Командний рядок може бути записаний в одній з чотирьох форм:

```
[позначка:] директива [операнди] [коментарі]
[позначка:] команда [операнди] [коментарі]
[позначка:] коментарі
порожній рядок
```

Коментар має наступну форму:

```
; текст коментаря
```

1.2. Вирази асемблера

Програми на асемблері містять вирази, які можуть складатися з операндів, функцій і операторів. Всі вирази в пам'яті займають по 4 байти. Операндами виразів можуть бути:

- числа (шістнадцяткові, десяткові, вісімкові і двійкові (шістнадцяткові числа мають признак 0x або $ (0x1a, 0xff, $ff), десяткові числа не мають признаків (255, 0), вісімкові числа починаються з 0 (03, 05), двійкові числа мають признак 0b (0b00001111, 0b11111111)).

- позначки (label:);

- коди символів ASCII ('A') і символьні рядки з ASCII кодів ("Hello asm");

- імена констант (визначені директивою .EQU), змінних (визначені директивою .SET), регістрів (визначені директивою .DEF);

- поточне значення лічильника команд ($) .

При запису виразів можна використовувати арифметичні і логічні операції, та операції відношень, табл. 1.

Таблиця 1 – Операції асемблера

Символ	Описання	Символ	Описання
!	Логічне НІ	<=	Менше або дорівнює
~	Бітове НІ	>	Більше
-	Унарний мінус	>=	Більше або дорівнює
*	Множення	==	Дорівнює
/	Ділення	!=	Не дорівнює
+	Додавання	&	Бітове І (And)
-	Віднімання	^	Виключне бітове АБО (Xor)
<<	Зсув вліво	|	Бітове АБО (Or)
>>	Зсув вправо	&&	Логічне І (And)
<	Менше	||	Логічне АБО (Or)

Групу арифметичних операцій утворюють додавання двох чисел або виразів (N+M), віднімання (N-M), множення (N*M), ділення (N/M), зміна знаку числа (-N).

Групу логічних операцій утворюють інверсія (~N), побітове І (N&M), побітове АБО (N|M), побітове виключне АБО (N^M), зсув вліво (N<<M – зсунути N вліво на M розрядів), зсув вправо (N>>M – зсунути N вправо на M розрядів). Операції відношень:

- логічне заперечення (!N – повертає 1, якщо N = 0, і – 0, якщо N не 0);
- менше (N < M – повертає 1, якщо N < M, і 0, якщо N > M);
- більше (N > M – повертає 1, якщо вираз N > M, і 0, якщо N < M);
- менше або дорівнює (N <= M – повертає 1, якщо N <= M, і 0, якщо N > M);
- більше або дорівнює (N >= M – повертає 1, якщо N >= M, и 0, якщо N < M);
- дорівнює (N = M – повертає 1, якщо N = M, і 0, якщо N ≠ M);
- не дорівнює (N != M - повертає 1, якщо N ≠ M, и 0, якщо N = M);
- логічне І (N&&M – повертає 1, якщо N ≠ 0 s M ≠ 0, інакше 0);
- логічне АБО (N||M – повертає 0, якщо N = 0 і M = 0, інакше 1).

Для задання черговості операцій використовуються круглі дужки.

1.3. Директиви асемблера

При написанні програм на мові асемблера використовують директиви. Директиви не транслюються безпосередньо в інструкції, а вказують компілятору положення програми в пам’яті, визначають макроси, ініціалізують пам’ять і т. і. Всі директиви починаються з крапки.

Великі за розміром програми можна розбивати на підпрограми. Частини програм, які часто повторюються, але не повинні виконуватися як підпрограми, записують як макровизначення (макроси) директивами .MACRO, .ENDMACRO. При виконанні макросу макроімена замінюються фактичними параметри. Макроімен в макросі може бути до 10 і позначаються вони як @0,…,@9. При виклику макросу фактичні параметри розділяються комами і ставляться у відповідність до макроімен:

```
.MACRO SUBI16              ; Початок макровизначення
```

```
    subi @1,low(@0)        ; Віднімання молодших байтів
    sbci @2,high(@0)       ; Віднімання старших байтів
.ENDMACRO                 ; Кінець макровизначення
.CSEG                     ; Початок сегмента коду
    ldi r16,0x55
    ldi r17,0x55
    SUBI16 0x1234,r16,r17 ;Відняти @0=0x1234 від @1=r17:@2=r16 => 0x43:0x21
```

Таблиця 2 – Директиви

Директиви	Описання	
BYTE	Резервування N байтів в SRAM або EEPROM	.equ N 32 table: .byte N
CSEG	Сегмент коду	
CSEGSIZE	Розмір пам’яті програм FLASH	
DB	Розміщення байту (байтів) в пам'яті програм FLASH або EEPROM	`consts: .db 0, 255, $0A, -4, 0b01010101, -128`
DEF	Призначення регістру додаткового імені	.def temp = r16
DEVICE	Задання асемблеру моделі МК	.device atmega8515
DSEG	Сегмент даних в SRAM	
DW, DD, .DQ	Розміщення слова (2-х слів, 4-х слів) в пам'яті програм або EEPROM	Const: .dw -3,100, $50
ENDM, ENDMACRO	Кінець макровизначення	
EQU	Визначити константну в обл. SRAM	.equ var1 = 100
ESEG	Сегмент констант EEPROM	
EXIT	Вийти з файлу	
INCLUDE	Прочитати asm код із вказаного файлу	.include “iodefs.asm”
LIST	Включити генерацію роздруку програми	
LISTMAC	Включити вставлення макророзширень у файл роздруку	
MACRO	Початок макророзширення	.macro subi16
NOLIST	Виключити генерацію роздруку програми	
ORG	Встановити адресу початку сегменту даних, констант, програми)	.org $100
SET	Визначити змінну в обл. SRAM	.set a=0x1a .set a=a+1

1.4. Система команд

Для програмування МК ATmegs8/ATmega48/ATmega8515/ATmega8535 використовується команди асемблера, які можна розділити на групи:

- арифметичні і логічні;
- пересилання і завантаження;
- передачі управління;
- роботи з бітами.

Нижче в таблицях вказано набір команд МК ATmegs8 / ATmega48 / ATmega8515 / ATmega8535 з наступними позначеннями:

Rd, *Rr* - регістри загального призначення з номерами *d* і *r*;

Rdh:*Rdl* - пара регістрів;

K - константа (дані);

P,*b* - розряд *b* (*b* = 0, ...,7) порту *P*;

Rr(b) - розряд *b* (*b* = 0, ...,7) регістра *Rr*;

(*X*), (*Y*), (*Z*) – вміст комірок, які адресуються регістровими парами *X*, *Y*, *Z* відповідно;

n - номер біта;

s - номер розряду в регістрі *SREG*;

PC – вміст програмного лічильника;

k – приріст в лічильнику команд;

q – 6-розрядне зміщення;

STACK – область пам'яті SRAM, яка адресується вказівником стека SP;

C, *Z*, *N*, *V*, *S*, *H*, *T*, *I* – біти регістра стану SREG;

d, *r* = 0.. .31 у всіх випадках, крім спеціально позначених.

Таблиця 3 – Арифметичні і логічні команди

Мнемо-ніка	Операн-ди	Описання	Функція	Прапори	Цикли
ADD	Rd,Rr	Додавання з без перенесенням	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Додавання з перенесенням	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Додавання константи до слова	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Віднімання без перенесення	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Віднімання константи	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Віднімання з перенесенням	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Віднімання константи з перенесенням	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логічне І (AND)	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне І (AND) з константою	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Логічне АБО (OR)	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логічне АБО (OR) з константою	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логічне виключне АБО (OR)	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Доповнення до 1	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Доповнення до 2	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Встановити біт(и) в регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Очистити біт(и) в регістрі	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Інкремент регістра	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декремент регістра	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Тест на нуль або негативне	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Очистити регістр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Встановити регістр	Rd = $FF		1
SBIW	Rdl,K6	Відняти константу (0-63) від слова	Rdh:Rdl = Rdh:Rdl - K6	Z,C,N,V,S	2
MUL	Rd,Rr	Множення беззнакове	R1:R0 = Rd * Rr	Z,C	2

MULS	Rd,Rr	Множення знакове	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Множення знакового і без знакового числа	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Множення беззнакових 8-бітових чисел з дробовою частиною із зсувом результату на 1 розряд вліво	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Множення знакових 8-бітових чисел з дробовою частиною із зсувом результату на 1 розряд вліво	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Множення знакового 8-бітового числа з дробовою частиною на 8-бітів беззнакове число з дробовою частиною із зсувом результату на 1 розряд вліво	R1:R0 = (Rd * Rr) << 1	Z,C	2

Таблиця 4 – Команди пересилання і завантаження даних

Мнемоніка	Операнди	Описання	Функція	Прапори	Цикли
MOV	Rd,Rr	Скопіювати регістр	Rd = Rr		1
MOVW	Rd,Rr	Скопіювати регістрову пару	Rd+1:Rd = Rr+1:Rr, r,d even		1
LDI	Rd,K8	Завантажити константу в регістр	Rd = K		1
LDS	Rd,k	Завантажити дані з адресу пам’яті	Rd = (k)		2*
LD	Rd,X	Завантажити дані з пам’яті, адрес якої міститься в регістрі вказівнику X, у регістр Rd	Rd = (X)		2*
LD	Rd,X+	Завантажити дані з пам’яті, адрес якої задано в регістрі X, в Rd та інкрементувати X	Rd = (X), X=X+1		2*
LD	Rd,-X	Декрементувати X та завантажити дані з пам’яті, адрес якої міститься регістрі X, в Rd	X=X-1, Rd = (X)		2*
LD	Rd,Y	Завантажити дані з пам’яті, адрес якої міститься в регістрі вказівнику Y, у регістр Rd	Rd = (Y)		2*
LD	Rd,Y+	Завантажити дані з пам’яті, адрес якої задано в регістрі Y, в Rd та інкрементувати Y	Rd = (Y), Y=Y+1		2*
LD	Rd,-Y	Декрементувати Y та завантажити дані з пам’яті, адрес якої міститься регістрі Y, в Rd	Y=Y-1, Rd = (Y)		2*
LDD	Rd,Y+q	Завантажити дані з пам’яті, адрес якої задано в регістрі Y із зміщенням q (значення регістрової пара не змінюється)	Rd = (Y+q)		2*
LD	Rd,Z	Завантажити дані з пам’яті, адрес якої міститься в регістрі вказівнику Z, у регістр Rd	Rd = (Z)		2*
LD	Rd,Z+	Завантажити дані з пам’яті, адрес якої задано в регістрі Z, в Rd та інкрементувати Z	Rd = (Z), Z=Z+1		2*
LD	Rd,-Z	Декрементувати Z та завантажити дані з пам’яті, адрес якої міститься регістрі Z, в Rd	Z=Z-1, Rd = (Z)		2*
LDD	Rd,Z+q	Завантажити дані з пам’яті, адрес якої задано в регістрі Z із зміщенням q (значення регістрової	Rd = (Z+q)		2*

		пара не змінюється)			
STS	k,Rr	Прямий запис у пам’ять з регістра Rr	(k) = Rr		2*
ST	X,Rr	Непрямий запис у пам’ять з регістра Rr	(X) = Rr		2*
ST	X+,Rr	Непрямий запис у пам’ять з постінкрементом	(X) = Rr, X=X+1		2*
ST	-X,Rr	Непрямий запис у пам’ять з преінкрементом	X=X-1, (X)=Rr		2*
ST	Y,Rr	Непрямий запис у пам’ять з регістра Rr	(Y) = Rr		2*
ST	Y+,Rr	Непрямий запис у пам’ять з постінкрементом	(Y) = Rr, Y=Y+1		2
ST	-Y,Rr	Непрямий запис у пам’ять з преінкрементом	Y=Y-1, (Y) = Rr		2
STD	Y+q,Rr	Непрямий запис у пам'ять із зміщенням (значення регістрової пара не змінюється)	(Y+q) = Rr		2
ST	Z,Rr	Непрямий запис у пам'ять із зміщенням	(Z) = Rr		2
ST	Z+,Rr	Непрямий запис у пам’ять з постінкрементом	(Z) = Rr, Z=Z+1		2
ST	-Z,Rr	Непрямий запис у пам’ять з преінкрементом	Z=Z-1, (Z) = Rr		2
STD	Z+q,Rr	Непрямий запис у пам'ять із зміщенням (значення регістрової пара не змінюється)	(Z+q) = Rr		2
LPM	None	Завантажити з пам’яті програм у R0	R0 = (Z)		3
LPM	Rd,Z	Завантажити з пам’яті програм у Rd	Rd = (Z)		3
LPM	Rd,Z+	Завантажити з пам’яті програм у Rd з постінкрементом	Rd = (Z), Z=Z+1		3
SPM	None	Записати у пам'ять програм	(Z) = R1:R0		-
IN	Rd,P	Завантажити з порту у регістр	Rd = P		1
OUT	P,Rr	Записати з регістра у порт	P = Rr		1
PUSH	Rr	Записати вміст регістра в стек	STACK = Rr		2
POP	Rd	Завантажити регістр Rd із стеку	Rd = STACK		2

Таблиця 5 – Команди передачі управління

Мнемо-ніка	Опе-ранд	Описання	Функція	Прапори	Цикли
RJMP	k	Відносний перехід	PC = PC + k +1		2
IJMP		Непрямий перехід за адресою регістра (Z)	PC = Z		2
JMP	k	Перехід за адресою k	PC = k		3
RCALL	k	Відносний виклик підпрограми	STACK = PC+1, PC = PC + k + 1		3/4*
ICALL		Непрямий виклик підпрограми за адресою (Z)	STACK = PC+1, PC = Z		3/4*
CALL	k	Виклик підпрограми	STACK = PC+2, PC = k		4/5*
RET		Повернення з підпрограми	PC = STACK		4/5*
RETI		Повернення з переривання	PC = STACK	I	4/5*
CPSE	Rd,Rr	Порівняти Rd і Rr, якщо вони однакові пропустити наступну інструкцію	if (Rd ==Rr) PC = PC 2 or 3		1/2/3
CP	Rd,Rr	Порівняти регістри Rd, Rr	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Порівняти регістри Rd, Rr з врахуванням перенесення	Rd - Rr - C	Z,C,N,V,H,S	1

CPI	Rd,K8	Порівняти регістр з константою	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Якщо біт b в рег. Rr очищений, то пропустити наступну інструкцію	if(Rr(b)==0) PC = PC + 2 or 3		1/2/3
SBRS	Rr,b	Якщо біт b в рег. Rr встановлений, то пропустити наступну інструкцію	if(Rr(b)==1) PC = PC + 2 or 3		1/2/3
SBIC	P,b	Якщо біт b порту P очищений, то пропустити наступну інструкцію	if(I/O(P,b)==0) PC = PC + 2 or 3		1/2/3
SBIS	P,b	Якщо біт b порту P встановлений, то пропустити наступну інструкцію	if(I/O(P,b)==1) PC = PC + 2 or 3		1/2/3
BRBC	s,k	Відносний перехід на k, якщо біт k в рег. SREG очищений	if(SREG(s)==0) PC = PC + k + 1		1/2
BRBS	s,k	Відносний перехід на k, якщо біт k в рег. SREG встановлений	if(SREG(s)==1) PC = PC + k + 1		1/2
BREQ	k	Відносний перехід на k (за рівністю), якщо біт Zero в рег. SREG встановлений	if(Z==1) PC = PC + k + 1		1/2
BRNE	k	Відносний перехід на k за (нерівністю), якщо біт Zero в рег. SREG очищений	if(Z==0) PC = PC + k + 1		1/2
BRCS	k	Відносний перехід на k, якщо біт Carry в рег. SREG встановлений	if(C==1) PC = PC + k + 1		1/2
BRCC	k	Відносний перехід на k, якщо біт Carry в рег. SREG очищений	if(C==0) PC = PC + k + 1		1/2
BRSH	k	Відносний перехід на k, якщо результат порівняння Rd<=Rs	if(C==0) PC = PC + k + 1		1/2
BRLO	k	Відносний перехід на k, якщо результат порівняння Rd<Rs	if(C==1) PC = PC + k + 1		1/2
BRMI	k	Відносний перехід на k, якщо ,біт Neg в рег. SREG встановлений	if(N==1) PC = PC + k + 1		1/2
BRPL	k	Відносний перехід на k, якщо ,біт Neg в рег. SREG очищений	if(N==0) PC = PC + k + 1		1/2
BRGE	k	Відносний перехід на k, якщо результат порівняння Rd>=Rs	if(S==0) PC = PC + k + 1		1/2
BRLT	k	Відносний перехід на k, якщо результат порівняння Rd<Rs	if(S==1) PC = PC + k + 1		1/2
BRHS	k	Відносний перехід на k, якщо Half carry біт встановлений	if(H==1) PC = PC + k + 1		1/2
BRHC	k	Відносний перехід на k, якщо Half carry біт очищений	if(H==0) PC = PC + k + 1		1/2
BRTS	k	Відносний перехід на k, якщо біт T встановлений	if(T==1) PC = PC + k + 1		1/2
BRTC	k	Відносний перехід на k, якщо біт T очищений	if(T==0) PC = PC + k + 1		1/2
BRVS	k	Відносний перехід на k, якщо біт Overrflow встановлений	if(V==1) PC = PC + k + 1		1/2

BRVC	k	Відносний перехід на k, якщо біт Overrflow очищений	if(V==0) PC = PC + k + 1		1/2
BRIE	k	Відносний перехід на k, якщо біт Interrupt - дозволено	if(I==1) PC = PC + k + 1		1/2
BRID	k	Відносний перехід на k, якщо біт Interrupt -заборонено	if(I==0) PC = PC + k + 1		1/2

Таблиця 6 – Команди роботи з бітами

Мне-моні-ка	Опе-ранд	Описання	Функція	Прапори	Цик ли
LSL	Rd	Логічний зсув вліво із встановленням перенесення	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H, S	1
LSR	Rd	Логічний зсув вправо із встановленням перенесення	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Логічний зсув вліво через біт перенесення	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H, S	1
ROR	Rd	Логічний зсув вправо через біт перенесення	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Арифметичний зсув вправо	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Переставлення тетрод	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)		1
BSET	s	Встановити біт S в SREG	SREG(s) = 1	SREG(s)	1
BCLR	s	Очистити біт S в SREG	SREG(s) = 0	SREG(s)	1
SBI	P,b	Встановити біт b в регістрі I/O	I/O(P,b) = 1		2
CBI	P,b	Очистити біт b в регістрі I/O	I/O(P,b) = 0		2
BST	Rr,b	Записати біт b з регістра Rr у біт T рег. SREG	T = Rr(b)	T	1
BLD	Rd,b	Скопіювати біт T у біт b рег. Rd	Rd(b) = T		1
SEC		Встановити біт Carry	C =1	C	1
CLC		Очистити біт Carry	C = 0	C	1
SEN		Встановити біт Negative	N = 1	N	1
CLN		Очистити біт Negative	N = 0	N	1
SEZ		Встановити біт Zero	Z = 1	Z	1
CLZ		Очистити біт Zero	Z = 0	Z	1
SEI		Встановити біт Interrupt	I = 1	I	1
CLI		Очистити біт Interrupt	I = 0	I	1
SES		Встановити біт Signed	S = 1	S	1
CLS		Очистити біт Signed	S = 0	S	1
SEV		Встановити біт Overflow	V = 1	V	1
CLV		Очистити біт Overflow	V = 0	V	1
SET		Встановити біт T	T = 1	T	1
CLT		Очистити біт T	T = 0	T	1
SEH		Встановити біт Half carry	H = 1	H	1
CLH		Очистити біт Half carry	H = 0	H	1

NOP		Немає операції			1
SLEEP		Перейти в “сплячий” режим			1
WDR		Скинути сторожовий таймер			1

2. Практична частина

2.1. Інтегроване середовище Atmel Studio

Atmel Studio – це інтегроване середовище розробки додатків для МК фірми Microchip. Версія Atmel Studio 7 об’єднує засоби керування проектами, текстовий редактор, компілятор C/C++, асемблер і налагоджувач програм. Таким чином, Atmel Studio підтримує проектування на стадіях розробки, налагодження і верифікації програмного забезпечення. Крім того, Atmel Studio підтримує цілий ряд апаратних платформ, які дозволяють програмувати МК, і внутрішньосхемні емулятори ICE40, ICE50, ICE200, JTAG ICE. Atmel Studio поширюється безкоштовно і доступна на сайті фірми Microchip http://www.microchip.com.

Для створення нового проекту потрібно вибрати пункт меню *Project/New project*. В наступному вікні задати мову програмування (асемблер або С), ім’я і розміщення проекту та натиснути кнопку OK, рис. 1:

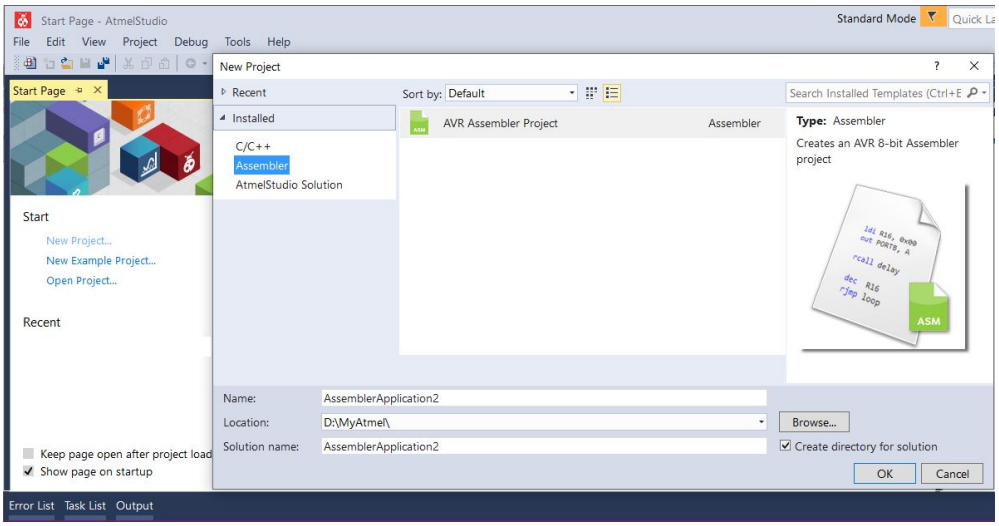


Рисунок 1 – Вибір мови програмування, назви і розміщення проекту

В новому вікні вибору пристроїв вибрати МК для якого є симулятор, а також потрібна плата для налагодження програми і прошивки МК, та натиснути ОК. В результаті буде створений порожній проект і каркас програми *main.asm*, рис. 2:

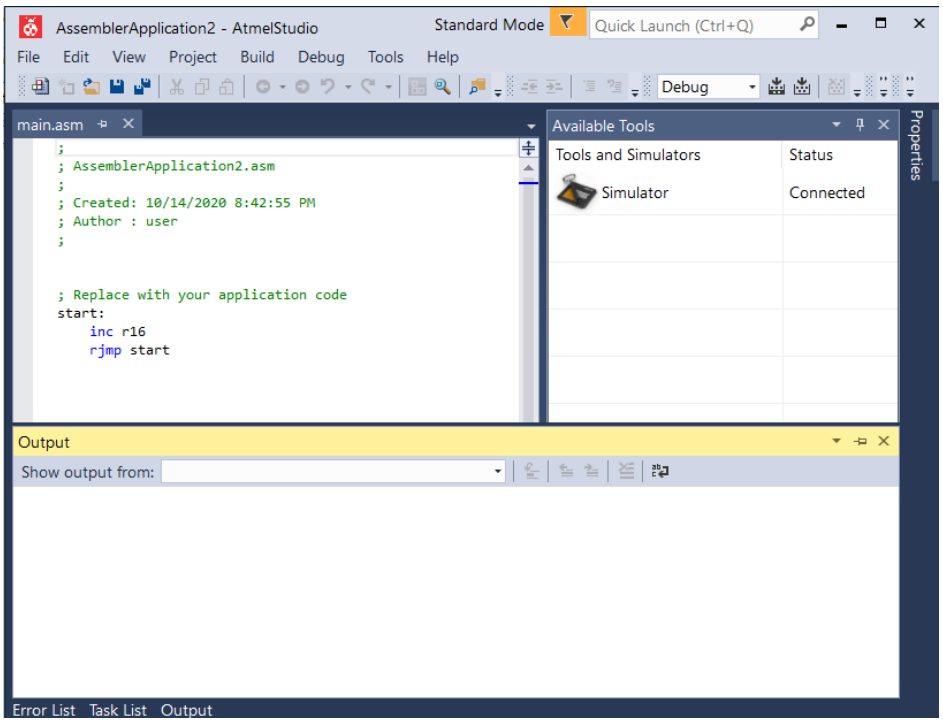


Рисунок 2 – Вибір моделі МК

При необхідності в проекті можна поміняти тип МК або змінити симулятор на прошивочну або налагоджувальну плату з фізичним МК, рис. 3.

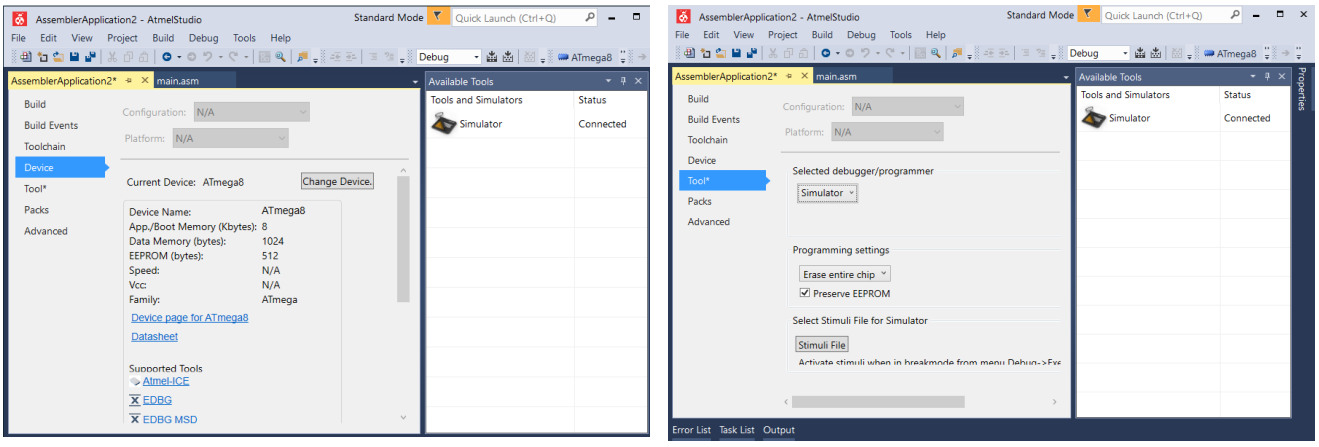


Рисунок 3 — Зміна типу МК або симулятора

В основній програмі записується асемблерний код, наприклад додати 2+5.

```
ldi r16,2
mov r0,r16
ldi r16,5
mov r1,r16
mov r2,r0
add r2,r1
```

Для трансляції і компонування програми вибирається пункт меню *Build/Build <aplication>* (перед цим необхідно вказати яку програму потрібно отримати — debug (для налагодження) / release (для виконання). Якщо трансляція пройшла без помилок, то виводиться відповідне повідомлення, позначене зеленим кружком і створюються файли з розширенням *.obj і *.hex.

Перевірити роботу алгоритму можна за допомогою симулятора (Simulator). В програмі потрібно вибрати досліджувані інструкції і позначити в них точку зупинки *Debug/Toggle Breakpoint*. Запустити на виконання програму в симуляторі можна вибором *Debug/Start debug and break*.

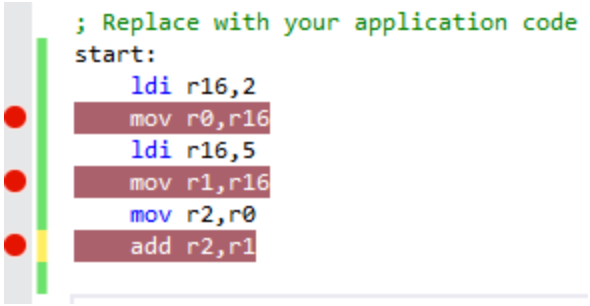


Рисунок 4 — Інструкції, позначені точками зупинки

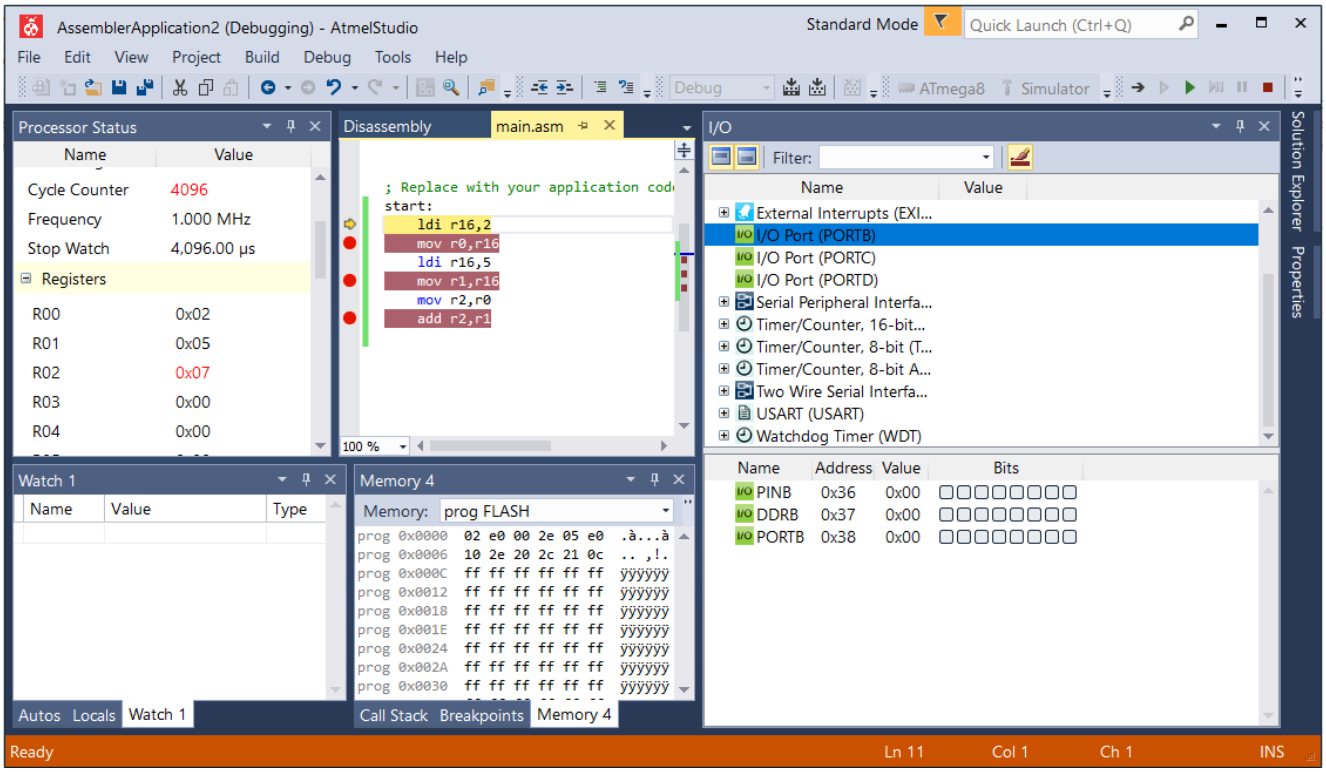


Рисунок 5 – Графічне вікно проекту і налагоджувальна програма

Завдання.

1. Запустити середовище Microchip Studio або Atmel Studio і навчитись створювати новий проект.

2. Ознайомитися з набором команд асемблера конкретної моделі МК, який приводиться в технічній документації виробника.

3. Написати програму на асемблері МК ATmegs8/ATmega48/ATmega8515/ATmega8535 або іншої моделі, яка обчислює значення виразів y, для a=40+k, b=30+k, c=20+k, d=10+k і записує значення a, b, c, d, у пам'ять даних SRAM починаючи з адреси ADR. Значення констант a,b,c,d визначити у пам’яті програм Flash.

Таблиця 7 – Варіанти завдань

Номер варіанту	**Вирази y**	**ADR** адреса початку даних в SRAM	**k**
1	y=a*b+c/2+d	071	3
2	y=a-b/2+c*d	062	4
3	y=a*b+(c-d)	073	5
4	y=a*b*c/d	064	3
5	y=(a+b)*(b+c)+d/4	085	4
6	y=d-a/2+b/2+c/2	076	5
7	y=a-b*c+d/4	067	3
8	y=(a-c/2)*(b-d/4)	078	4
9	y=a*b*c*d/8	069	5
10	y=a*(b+c/2+d)	06A	3
11	y=a*(b-c/2)+d	06B	4
12	y=(a-b*c-d)/2	07C	5
13	y=(a*b+c/2)*d	08D	3
14	y=a*b*c+d/2	07E	4
15	y=a/4+b/2+c/8+d/16	071	5
16	y=a/2+b*2+c/4+d*4	06E	3
17	y=(a+b+c*d)/2	062	4
18	y=a+(b-c-d)/4	07	5
19	y=(a+b*c)/4+d	07C	3
20	y=(a*b+c/2)*d	076	4
21	y=(a*b*c+d)8	08B	5
22	y=a*b*c-d/2	08C	3
23	y=a/2+b/4-c/8+d*16	08D	4
24	y=(a*16+ b*16+c*16+d*16	08E	5
25	y=(a+2)*(b-3)*(c+4)*d/8	077	3

Примітка. Номер варіанту завдання вибирається за порядковим номером студента у журналі групи.

Зміст звіту:

1. Титульна сторінка.
2. Назва, мета та завдання до лабораторної роботи.
3. Короткий опис теоретичної частини.
4. Блок схема алгоритму згідно ДСТУ.

5. Текст програми із коментаріями.
6. Вміст регістрів і SRAM після виконання програми.

Приклад виконання завдання

У пам’яті програми розміщено N = $0A ($10_{10}$) кодів, наприклад: $11, $22, $33, $44, $55, $66, $77, $88, $99, $AA. Необхідно переписати їх в оперативну пам’ять даних SRAM, починаючи з адреси ADR_data = $064.

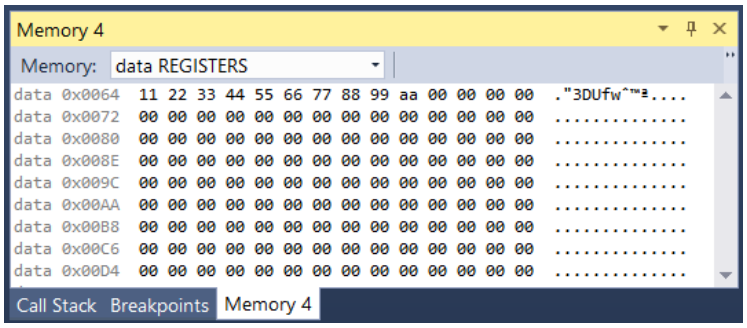
Текст програми:

```
; Replace with your application code

.SET ADR_data = $064  ; стартова адреса даних (SRAM=>$060)
.SET N = $0A          ; довжина вектора

.DSEG
.ORG ADR_data
 data: .BYTE N       ; резервування N байт пам'яті SRAM

.CSEG
const: .DB  $11, $22, $33, $44, $55, $66, $77, $88, $99, $AA

start:
LDI R16,N       ;запис у регістр R16 (r16-r31) лічильника чисел N

LDI R31,0       ;чистка вмісту верхньої частки регістрової пари  Z (R31:R30)
LDI R30,const   ; завантаження адреси констант (0, 1, ..., 0x0A)

LDI R29,0       ;чистка вмісту верхньої частки регістрової пари  Y (R29:R28)
LDI R28,ADR_data ;запис у регістрову  пару Y адреси  data

L1:
LPM R3,Z     ; прочитати константу за адресою рег. Z у R3
ST  Y,R3     ;записати число з R3 за адресою, яка знаходиться в Y

DEC R16      ;зменшити зміст лічильника на 1
INC R30      ;збільшити адресу Z на 1  (констант)
INC R28      ;збільшити адресу Y на 1  (даних в SRAM)
CPI R16,0    ;порівняти вміст R0 з 0
BRNE L1      ;якщо R0 не 0 то перейти на L1
rjmp start
```

Вміст регістрів після виконання програми

Name	Value
R03	0xAA
R19	0xBD
R20	0x00
R21	0x65
R28	0x6E
R29	0x00
R30	0x0A
R31	0x00

Вміст SRAM після виконання програми.



Питання.

1. Чим відрізняється мова асемблер від мов програмування високого рівня?
2. Яка структура програми асемблера її елементи?
3. З чого складаються вирази асемблера?
4. Які є директиви асемблера і їх призначення?
5. Які є команди асемблера і на які групи вони поділяються?
6. Розказати про основні арифметичні і логічні команди.
7. Розказати про основні команди пересилання і завантаження даних.
8. Розказати про основні команди передачі управління.
9. Розказати про основні команди роботи з бітами.
10. Розказати про способи адресації пам’яті даних.
11. Розказати про способи адресації пам’яті програм.
12. Розказати як створити новий проект в графічному середовищі.

ЛАБОРАТОРНА РОБОТА № 3. Арифметичні операції з двійковими числами

Мета роботи – вивчення способів подання двійкових чисел в МК AVR, алгоритмів арифметичних операцій, програмування арифметичних процедур.

1. Основні теоретичні відомості

Числа в МК подаються у вигляді цілих (крапка після молодшого розряду) або у вигляді дробових (крапка перед старшим розрядом) без знаку (0…255) або із знаком. Для цілих і дробових чисел знак займає старший розряд : 0 – плюс, **1** – мінус (1xxx, .**1**xxx). Діапазон позитивних чисел із знаком 0…127, негативних 0…-128 (з врахуванням біту перенесення Carry).

Негативні цілі і дробові числа подаються у доповняльному коді. Для цілих $[A]_{доп} = 2^{k+1} - |A|$, для дробових $[A]_{доп} = 2 - |A|$, де k – число числових розрядів. Для цілих це доповнення до основи числення 2^{k+1}, а для дробових – до 2. Наприклад, для -1 $[A]_{доп}=2^{7+1}-1= 256-1=255$ (1111_1111), для -128 $[A]_{доп}=2^{7+1}-128= 256-128=128$ (1000_0000). Якщо з такого подання числа, наприклад 240 (1111_0000), необхідно отримати негативне число, то потрібно із значення числа відняти 255 і добавити 1 (що рівносильне відніманню із 256). 240-256=-16. Результат такого віднімання без знаку називається доповненням до 2 (доповняльним кодом для числа 240). Для числа 16 доповняльним кодом буде число 240.

Доповняльний код можна отримати з оберненого коду додаючи до наймолодшого розряду один біт. Обернений (інверсний) код від'ємного числа отримують з його прямого коду після інвертування значень розрядів цифрової частини (знаковий розряд не міняється), тобто заміною нулів на одиниці і одиниць – на нулі.

Приклад1. Віднімання числа А можна замінити додаванням числа (2^{7+1}-А =256-А)

1. 5 – 7 = 5 + (256 -7) = 5 + 249 = 254
   254 – 256 = -2.
2. 7 – 5 = 7 + (256 -5) = 7 + 251 = 258
   258 – 256 = 2.

```
start:
   .cseg
 ; інверсний і доповняльний код
 ldi r16,7
 mov r0,r16
 com r0       ; 0xf8=інверсний код (доп. до 1)
 ldi r16,0b00000001
 add r0,r16   ; 0xf9=інв. код. + 1 біт = доповняльний код
 ldi r16,7
 mov r1,r16
 neg r1       ;  0xf9=доповняльний код (доп. до 2)
 ;  віднімання цілих чисел 5-7
 ldi r16,5    ; 0x05
 mov r1,r16
 ldi r16,-7   ; 0xf9=доповняльний код
 mov r2,r16   ; 0xf9=249
 add r1,r2    ; 0xfe=254
 ; віднімання цілих чисел7-5
 ldi r16,7
 mov r1,r16   ; 0x07
 ldi r17,-5   ; 0xfb
 mov r2,r17
```

```
add r1,r2    ; 0x02
```

Якщо при додаванні або відніманні отримують результат, який виходить за діапазон значень -128…127, то фіксується переповнення.

Правило виявлення переповнення.

При додаванні переповнення виникає тоді, коли доданки мають однакові знаки, а знак суми відрізняється від знаку доданків.

При відніманні переповнення виникає тоді, коли операнди мають різні знаки, а знак різниці відрізняється від знаку зменшуваного. Узагальнене правило: переповнення виникає тоді, коли перенесення із знакового розряду p8 і в знаковий розряд p7 різні – p8 XOR p7.

Розміри операндів простого додавання або віднімання обмежені значенням 255 для беззнакових чисел і -128÷127 для знакових чисел, що явно недостатньо для практики. Для реальних потреб числа можуть знаходитися в інтервалі 2^{16}=65 536 (двобайтові), 2^{24}=16,777,216 (трибайтові), 2^{32}=4,294,967,296 (чотирибайтові).

Так як МК AVR апаратні арифметичні операції цілочислові і 1-байтові , то для операндів з більшою кількістю байтів арифметичні операції реалізовуються програмно.

Приклад 2. Додавання 2-байтових чисел: R_3:R_2 + R_1:R_0

```
.cseg
.def RL1=r0
.def RH1=r1
.def RL2=r2
.def RH2=r3
.CSEG
ldi r16,0xc8   ; 200 => c8 => 1100_1000
mov RL1,r16
ldi r16,0xfa   ; 250 => fa => 1111_1010
mov RL2,r16
add RL1,RL2    ; 1100_0010  => пер.1 з с2 => пер.1 з 302
adc RH1,RH2    ; 1
               ; 0000_0001 1100_0010  = 1c2 = 450
```

Приклад 3. Додавання 4-байтових чисел R_7:R_6:R_5:R_4 + R_3:R_2:R_1:R_0

```
.def R1L=r0   ; 200
.def R1H=r1   ; 116
.def R2L=r2   ; 250
.def R2H=r3   ; 210
.def R3L=r4   ; 196
.def R3H=r5   ; 154
.def R4L=r6   ; 218
.def R4H=r7   ; 120

.cseg
ldi r16,0xc8   ; r0=200 = 0xc8
mov R1L,r16
ldi r16,0x74   ; r1=116 = 0x74
mov R1H,r16
ldi r16,0xfa   ; r2=250 = 0xfa
mov R2L,r16
ldi r16,0xd2   ; r3=210 = 0xd2
mov R2H,r16

ldi r16,0xc4   ; r4=196 = 0xc4
mov R3L,r16
ldi r16,0x9a   ; r5=154 = 0x9a
mov R3H,r16
ldi r16,0xda   ; r6=218 = 0xda
```

```
mov R4L,r16
ldi r16,0x78   ; r7=120 = 0x78
mov R4H,r16

add R3L,R1L    ; 196 + 200 = 0xc4 + 0xc8 = 0x8c пер.1
adc R3H,R1H    ; 154 + 116 = 0x9a + 0x74 = 0x0e + пер.1 = 0x0f
adc R4L,R2L    ; 218 + 250 = 0xda + 0xfa =  0xd4 + пер.1 = 0xd5
adc R4H,R2H    ; 120 + 210 =  0x78 + 0xd2 = 0x4a + пер.1 = 0x4b
```

2. Множення цілих чисел без знаку

Для множення двох 8-бітових чисел без знаку служить команда

```
mul Rd,Rs,    де  0  <= d <= 31, 0 <= s <= 31,
```

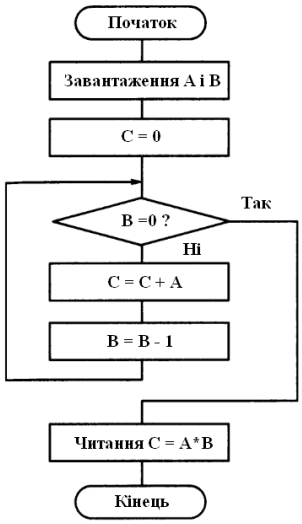
а результат множення розміщується в парі регістрів R_1:R_0.

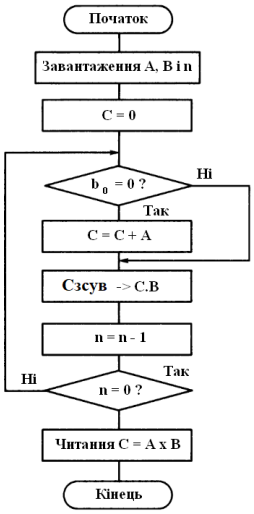
Множення цілих чисел без знаку можна виконати програмно за алгоритмом показаним на рис. 3.1, *а*. Множене А і множник В завантажуються в регістри загального призначення і обнуляється регістр добутку С. В циклі аналізується вміст регістра множника В і якщо В != 0, то до суми часткових добутків С додається множене А і зменшується на 1 множник. Якщо В = 0, то в С отримано добуток.

На практиці використовується метод множення цілих чисел додаванням ряду часткових добутків $C=\sum 2^i Ab_i$, де b_i- значення розряду множника. Один з алгоритмів множення, починаючи з молодших розрядів множника, із зсувом вправо суми часткових добутків, показано на рис.3.1, *б*.

Цей алгоритм можна використати для отримання добутку двох двійкових чисел без знаку. Кількість ітерацій множення визначається числом розрядів множника. Оскільки в процесі множення на кожній ітерації виконується зсув множника В на один розряд вправо, на місце звільненого розряду можна записати виштовхнутий при зсуві вправо розряд добутку С. Таким чином, 2*n*-розрядний добуток можна отримати, об'єднавши вміст *n*-розрядного регістра, в якому формується старша частина добутку, і регістра В, в якому після виконання множення залишиться молодша частина добутку.

а)





а) *б*)

Рисунок 1 – Схеми множення: *а*) проста; *б*) починаючи з молодших розрядів

Приклад 4. Множення і ділення двійкових чисел з каталогу AVR STUDIO /Atmel/ Avr tools/Avrassembler/Appnotes/avr200.asm:

```
; Replace with your application code
;**** A P P L I C A T I O N   N O T E   A V R 2 0 0 **********************
; atmega88
     rjmp  RESET ;очистити обробник

;***************************************************************************
;* "mpy8u" - 8x8 Bit Беззнакове множення
;* Примітка: Low byte множене і множник використовують спільний регістр.
;* В результаті множник буде перезаписаний результатом.
;***************************************************************************

;***** Регістрові змінні підпрограми

.def  mc8u  =r16       ;множене
.def  mp8u  =r17       ;множник
.def  m8uL  =r17       ;Low byte результату
.def  m8uH  =r18       ;High byte результату
.def  mcnt8u     =r19       ;лічильник циклу

;***** Код

mpy8u:     clr   m8uH  ;очистити High byte результату
     ldi   mcnt8u,8   ;ініціалізація лічильника
     lsr   mp8u       ;логічний зсув вліво множеного

m8u_1:     brcc  m8u_2 ;перевірка встановлення прапора carry
     add   m8uH,mc8u  ;додати множник до High byte результату
m8u_2:     ror   m8uH       ;циклічний зсув вправо High byte результату
     ror   m8uL       ;циклічний зсув вправо result Low byte результату і множника
     dec   mcnt8u           ;декремент лічильника
     brne  m8u_1      ;якщо Z !=0 продовжити цикл
     ret


;***************************************************************************
;* "mpy8s" - 8x8 Bit Знакове множення
;* Знакове множення двох регістрових змінних mp8s і mc8s.
;* Результат в регістрах m8sH, m8sL
;* Підпрограма реалізує алгоритм Booth's. Якщо потрібно всі 16 бітів
;* результату, уникати виклику підпрограми із значенням множника
;* -128 ($80)
;***************************************************************************

;***** Регістрові змінні підпрограми

.def  mc8s  =r16       ;множене
.def  mp8s  =r17       ;множник
.def  m8sL  =r17       ;Low byte результату
.def  m8sH  =r18       ;High byte результату
.def  mcnt8s     =r19       ;лічильник циклу

;***** Код

mpy8s:     sub   m8sH,m8sH  ;очистити High byte результату і carry
     ldi   mcnt8s,8   ;ініціалізація лічильника
m8s_1:     brcc  m8s_2 ;перевірка встановлення прапора
     add   m8sH,mc8s  ;   ;додати множник до High byte результату
m8s_2:     sbrc  mp8s,0     ;якщо поточний біт встановлений
     sub   m8sH,mc8s  ;відняти множник від High byte результату
     asr   m8sH       ;ариф. зсув вправо High byte результату
     ror   m8sL       ;цикл. зсув вправо Low byte результату і множника
     dec   mcnt8s           ;декремент лічильника
```

```
      brne  m8s_1 ;якщо Z !=0 продовжити цикл
      ret


;***** Основна програма

.def  temp  =r16       ;тимчасова змінна

;*****
RESET:
;-----------------------------------------------------------------
;Розкоментувати наступні рядки для МК з SRAM
      ldi   temp,low(RAMEND)
      out   SPL,temp
      ldi   temp,high(RAMEND)
      out   SPH,temp ;ініціалізація вказівника стеку (Stack Pointer)
;-----------------------------------------------------------------

;***** Множення двох беззнакових 8-Bit чисел (250 * 4)

      ldi   mc8u,250
      ldi   mp8u,4
      rcall mpy8u      ;результат: m8uH:m8uL = $03e8 (1000)

;***** Множення двох знакових 8-Bit чисел (-99 * 88)
      ;ldi  mc8s,-99
      ;ldi  mp8s,88
      ;rcall mpy8s     ;результат: m8sH:m8sL = $ddf8 (-8712)
```

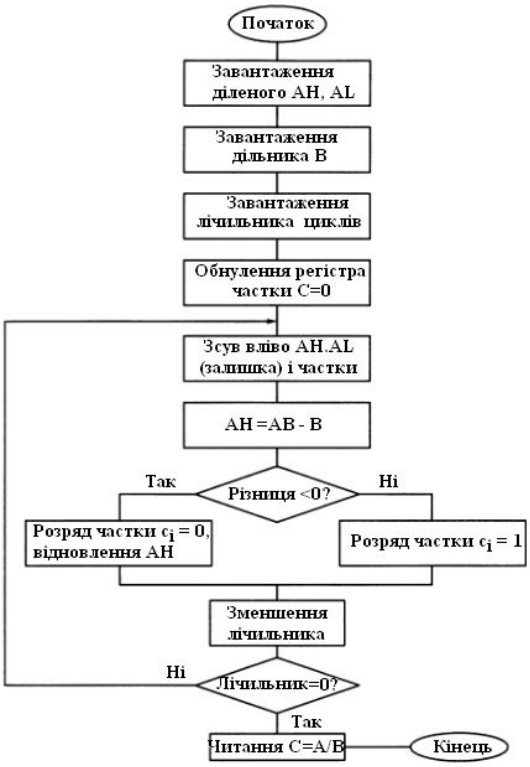
3. Ділення цілих чисел

Для типового алгоритму цілочисельного ділення C = A/B діленим є подвійне слово AH:AL (два байта), а дільником - одинарне B (один байт); частка C і залишок отримують як одинарні слова. При діленні необхідно запобігти можливості ділення на 0. Якщо для подання частки потрібно більш одного слова, то фіксується переповнення. При діленні необхідно перевірити умову - дільник має бути більшим старшого слова діленого (B > AH). При діленні цілих чисел можна використати алгоритм ділення без відновлення залишку і алгоритм з відновленням залишку. Схема алгоритму ділення з відновленням залишку зображена на рис. 2а.

Алгоритм ділення є ітераційною процедурою. На кожній ітерації спочатку подвоюється ділене (на першій ітерації) або залишок (на всіх наступних) шляхом зсуву вліво на один розряд, потім віднімається дільник і визначається цифра частки по знаку різниці. Якщо різниця позитивна, то визначена на даній ітерації цифра частки $c_i = 1$, якщо різниця негативна, цифра частки $c_i = 0$. Залишок відновлюється додаванням дільника до залишку після віднімання на поточній ітерації ділення. Ділення виконується до отримання всіх цифр частки.

Алгоритм ділення без відновлення залишку (рис. 2б) є ітераційною процедурою, на кожній ітерації якої або віднімається дільник B, що замінюється додаванням з доповняльним кодом $[-B]_{доп}$, або додається B в залежності від знаку залишку, отриманого на попередній ітерації ділення.

а)



б)

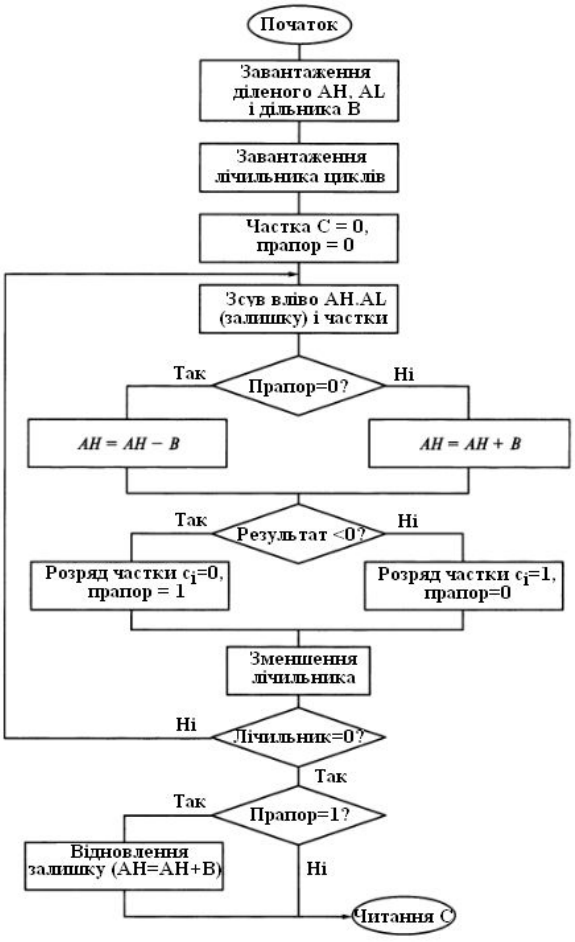


Рисунок 3 – Схема алгоритму ділення: *а* – з відновленням залишку; *б* – без відновленням залишку

Якщо отриманий залишок більший або дорівнює 0, при черговій ітерації ділення виконується віднімання B; якщо залишок менше 0 – додавання B. Перед кожним відніманням (або додаванням) залишок подвоюється шляхом зсуву вліво. На початковій ітерації ділення ділене зсовується на один розряд вліво. Ділення чисел із знаком можна виконати різними способами. Якщо початкові операнди задані в прямих кодах, то шляхом додавання по модулю 2 знакових розрядів можна визначити знак частки. Модулі діленого і дільника можна поділити, використовуючи один з вищевказаних алгоритмів. Для визначення переповнення потрібно виконати пробне віднімання $A - 2^{(k-1)} \cdot B$, резервуючи один розряд *k*-розрядної частки для знаку.

Приклад ділення 16-розрядного числа A на 8-розрядне число B з відновленням залишку:

A=1024=00000100.00000000 B=10=00001010

-B=$[-10]_{доп}$=11110110

C=c7c7c5c4c3c2c1c0 – частка x – біт, який вільно визначається при зсуві

00000100.00000000	ділене A (AH.AL)
+ 11110110	пробне віднімання B
11111010	так як різниця < 0, переповнення не має
000010000.0000000x	зсув вліво AH.AL
+11110110	віднімання B
11111110	1-й залишок менший 0, розряд частки c7 = 0
+11111100.000000xx	зсув вліво AH.AL
00001010	добавлення B
00000110	2-й залишок більший 0, розряд розряд частки c6 = 1
00001100.00000xxx	зсув вліво AH.AL
+11110110	віднімання B
00000010	3-й залишок більший 0, c5 = 1
00000100.0000xxxx	зсув вліво AH.AL
+11110110	віднімання B
11111010	4-й залишок менший 0, c4 = 0
11110100.000xxxxx	зсув вліво AH.AL
+00001010	додавання B
11111110	5-й залишок менший 0, c3 = 0
11111100.00xxxxxx	зсув вліво AH.AL
+00001010	додавання B
00000110	6-й залишок більший 0, c2 = 1
00001100.0xxxxxxx	зсув вліво AH.AL

+11110110	віднімання B
00000010	7-й залишок більший 0, c1 = 1
00000100.xxxxxxxx	зсув вліво AH.AL
+11110110	віднімання B
11111010	8-й залишок менший 0, c0 = 0
11111010	
+00001010	додати B
00000100	відновлений залишок AH = 4
	C= 01100110 = 102

Приклад 5. Добавити основну програму з прикладу 2б, закоментувати виклики підпрограми множення і розкоментувати – ділення.

```
; atmega88
rjmp  RESET ;очищення обробника
;***************************************************************************
;* "div8u" - 8/8 Bit беззнакове ділення
;* Ділення двох регістрових змінних "dd8u" (ділене) і "dv8u" (дільника).
;* Результат в "dres8u" і залишок в "drem8u".
;***************************************************************************
;***** Регістрові змінні підпрограми
.def drem8u     =r15  ;залишок
.def dres8u     =r16  ;результат
.def dd8u  =r16  ;ділене
.def dv8u  =r17  ;дільник
.def dcnt8u     =r18  ;лічильник циклів

;***** Код
div8u:     sub   drem8u,drem8u    ;очищення залишку і прапора carry
     ldi   dcnt8u,9   ;ініціалізація лічильника циклів
d8u_1:     rol   dd8u  ;зсув вліво діленого
     dec   dcnt8u     ;декремент лічильника
     brne  d8u_2 ;якщо лічильник не нуль
     ret         ;    return
d8u_2:     rol   drem8u ;цикл. зсув діленого в залишок
     sub   drem8u,dv8u      ;залишок = залишок - дільник
     brcc  d8u_3 ;якщо результат негативний
     add   drem8u,dv8u      ;відновити залишок
     clc              ;очистити carry для зсуву в результат
     rjmp  d8u_1 ;інакше
d8u_3:     sec         ;встановити carry для зсуву в результат
     rjmp  d8u_1

;***************************************************************************
;* "div8s" - 8/8 Bit знакове ділення
;* Ділення двох регістрових змінних "dd8s" (ділене) і "dv8s" (дільник)
;* Результат в "dres8s" і залишок в "drem8s".
;***************************************************************************
;***** Регістрові змінні підпрограми
.def d8s   =r14       ;знаковий регістр
.def drem8s     =r15  ;залишок
.def dres8s     =r16  ;результат
.def dd8s  =r16  ;ділене
.def dv8s  =r17  ;дільник
.def dcnt8s     =r18  ;лічильник циклів

;***** Код
```

```
div8s:     mov  d8s,dd8s   ;скопіювати ділене у знаковий регістр
      eor  d8s,dv8s        ;xor знак з дільником
      sbrc dv8s,7          ;якщо MSB дільника встановлений
      neg  dv8s            ;    змінити знак дільника
      sbrc dd8s,7          ;якщо MSB діленого встановлений
      neg  dd8s            ;    змінити знак дільника
      sub  drem8s,drem8s   ;очистити залишок і прапор carry
      ldi  dcnt8s,9        ;ініціювати лічильник циклів
d8s_1:     rol  dd8s       ;зсунути вліво ділене
      dec  dcnt8s          ;декремент лічильника
      brne d8s_2      ;якщо лічильник не нуль
      sbrc d8s,7      ;якщо MSB знакового регістра встановлений
      neg  dres8s     ;змінити знак результату
      ret             ; return
d8s_2:     rol  drem8s ;цикл. зсув діленого у залишок
      sub  drem8u,dv8s      ;залишок = залишок - дільник
      brcc d8s_3       ;якщо результат негативний
      add  drem8u,dv8s      ;відновити залишок
      clc              ; очистити carry для зсуву в результат
      rjmp d8s_1       ;інакше
d8s_3:     sec         ;встановити carry для зсуву в результат
      rjmp d8s_1

;***** Main Program Register variables
.def temp =r16       ;тимчасова змінна
;***** Код
RESET:
;-----------------------------------------------------------------
;Include these lines for devices with SRAM
      ldi  temp,low(RAMEND)
      out  SPL,temp
      ldi  temp,high(RAMEND)
      out  SPH,temp   ;ініціалізація Stack Pointer
;-----------------------------------------------------------------
;***** Divide Two Unsigned 8-Bit Numbers (100/3)
      ldi  dd8u,100
      ldi  dv8u,3
      rcall div8u      ;результат:     r16 => $21 (33)
                       ;залишок:  r15 => $01 (1)
                       ; дільник  r17 =  $03 (3)
;***** Divide Two Signed 8-Bit Numbers (-110/-11)
      ldi  dd8s,-110
      ldi  dv8s,-11
      rcall div8s      ;результат:     r16 => $0a (10)
                       ;залишок   r15 => $00 (0)
                       ; дільник  r17 =  $0b (11)
      forever:rjmp    forever
```

5. Операції з дробовими числами

Наприклад потрібно помножити 76*0,485=36,86. Спочатку вручну множиться 0,485*128(2^7)=62,08≈62=0x3e. Тепер початкові операнди в операції множення замінюються наступними 76*62/128≈36. Операція ділення на 128 реалізується циклічними зсувами вправо молодшого байту із врахуванням прапора Overflow при логічних зсувах вправо старшого байта.

Приклад 6. Операція множення з дробовими числами.

```
;****************************************
; Множення 76 x 0,485 = 36,86
;               0,484x128(2^7)=62,08=62=0x3e
; 76x62/128=36,81 => 36 => 0x24
;****************************************
;atmeg88
```

```
; регістрові змінні
.def count = r18

.cseg
     ldi r16,0x4c      ; 76=>0x4c
     ldi r17,0x3e      ; 62=>0x3e
     mul r16,r17       ; r1:r0 = 4712=0x12:0x68 результат
; ділення на 2:18 зсувом
     clr count
div16:
     lsr r1      ; логічний   зсув вправо на 1 біт, LSB->V
     ror r0      ; циклічний  зсув вправо на 1 біт, V->HSB
     inc count   ; інкремент лічильника
     cpi count,7 ; порівняння регістра з константою
     brne div16  ; якщо не дорівнює перейти на div16
                    ; результат r0=0x24=36
```

6. Апаратне множення чисел дробовому форматі 1.7

Формат «*n.q*» позначає двійкове дробове число з *n* розрядами зліва та *q* розрядами справа десяткової крапки.

Для беззнакового дробового числа у форматі 1.7 діапазон можливих значень лежить в межах [0, 2>. Для знакового дробового числа один розряд для цілого значення відведений під знак, тому діапазон можливих значень лежить в межах [-1, 1>.

Беззнакове			Знакове		
двійкове	десяткове	дробове 1.7	двійкове	десяткове	дробове 1.7
0000_0000	0	0.0	1000_0000	-128	-1.0
0000_0001	1	0.078125	1000_0001	-127	-0.9921875
0000_0010	2	0.015625	1000_0010	-126	-0.984375
...	...	...	...	...	...
1111_0101	253	1.9765625	1111_0101	-3	-0.0234375
1111_1110	254	1.984375	1111_1110	-2	-0.015625
1111_1110	255	1.9921875	1111_1111	-1	-0.0078125

Щоб отримати реальні значення чисел у форматі 1.7 / 1.15, необхідно їх десяткові цілочислові подання поділити на число 128 / 32768 (2^7 / 2^{15}).

При множенні чисел у форматі «*n.q*», наприклад *n1.q1*× *n2.q2*, кінцевий результат матиме подання (*n1*+*n2*).(*q1*+*q2*). Для того щоб отриманий результат був у зручному форматі, апаратне множення виконує над результатом порозрядний зсув вліво на 1 біт, після чого результат може бути заокруглений до 1.7.

$$1.7\times 1.7 = (2.14) <<1 = 1.15$$

Для спрощення формування дробових чисел у форматі 1.7 та 1.15 передбачені спеціальні функції для компілятора:

- Q7() – конвертує число з плаваючою крапкою до знакового формату 1.7 (для fmul/fmuls/ fmulsu команд);
- Q15() – конвертує число з плаваючою крапкою до знакового формату 1.15;

- Int() – повертає цілу частину числа з плаваючою крапкою (відкидає дробову частину);
- Frac() – повертає дробову частину числа з плаваючою крапкою;
- Abs() – повертає абсолютне значення з числа.

Оскільки функція Q7() формує лише знакове дробове число 1.7, то для отримання беззнакового числа за допомогою цієї функції формується дробова частину числа і додається 1 у старший розряд.

```
ldi r16, 0.5*128         ; 64=0x40=0100_0000
ldi r16,Q7(0.5)          ; 64=0x40=0100_0000, 64/128=0.5
 ldi r17,(1<<7 | Q7(0.5)) ; 192(-64)=0xC0=1100_0000, 192/128=1.5
 ldi r18,(1<<7 | Q7(0.4)) ; 179(-51)=0xB3=1011_0011, 179/128=1.3984375

 ldi r19, -0.5*128        ; 192(-64)=0xC0=1100_0000, -64/128=-0.5
 ldi r19,Q7(-0.5)         ; 192(-64)=0xC0=1100_0000
 ldi r20,Q7(-0.9)         ; 140(-116)=0x8C=1000_1100, -116/128=-0.90625
 ; беззнакове 1.7 x беззнакове 1.7
 fmul r16,r17   ; r1:r0=0x60:00=24576  24576/(2^8 * 2^7 = 2^15 = 32768) = 0.75
 fmul r17,r18   ; r1:r0=0x0C:0x80=3200 3200/32768=0.09765625 переповнення

 ; знакове 1.7 x знакове 1.7
 fmuls r19,r19  ; r1:r0=0x20:0x00=57344(-8192) = -8192/(2^15 = 32768) = -0.25

 ; знакове 1.7 x беззнакове 1.7
 fmulsu r19,r17 ; r1:r0=0xA0:0x00=40960(-24576) = -24576/(2^15 = 32768) = -0.75
 fmulsu r20,r17 ; r1:r0=0x52:0x00=20992(-44544) = -44544/(2^15 = 32768) = -1.359375
```

При множенні беззнакових дробових чисел потрібно слідкувати, щоб не було переповнення, тобто результат не був більшим за значення 2. Аналогічно, при множенні знакового на беззнаковий також потрібно слідкувати, щоб не було виходу за межі діапазону [-1, 1>.

Якщо ж потрібно множити дробові числа, що виходять за межі, вказані для формату 1.7, наприклад 2.5×7.125, тоді необхідно умовно перейти до іншого формату. Для даного випадку підходить беззнаковий формат 3.5 з діапазоном значень [0; 8>. У результаті множення беззнакових дробових чисел 3.5 отримується результат у форматі 6.10. Так як при множенні результат зсовується вліво на 1 розряд, то кінцевий формат буде 5.11.

При завантаженні у регістр числа з плаваючою крапкою беззнакового формату 3.5 воно множиться на 2^5=32 (кількість розрядів після крапки).

Для відображення результату множення у у форматі 5.11 у число з плаваючою крапкою необхідно отриманий результат поділити на 2^{11}=2048 (кількість розрядів після крапки).

```
                    ; 2.5 * 7.125 = 17.8125
ldi r16, 2.5*32    ;  80=0x50=0101_0000
ldi r17, 7.125*32  ; 228=0xE4=1110_0100
fmul r16,r17       ; r1:r0=0x8E:0x80=36480/2048 = 17.8125
```

Аналогічно множаться знакові дробові числа інших форматів 2.6, 3.5, 4.4, 5.3, 6.2, 7.1. Формат 8.0 задає цілі числа, для множення яких є команди цілочислового множення mul, muls, mulsu.

7. Практична частина

Завдання.

1. Відкомпілювати і проаналізувати в режимі DEBUG виконання програм наведених у прикладах.

2. Згідно варіанту табл. 7 реалізувати на асемблері заданий алгоритм. Продемонструвати його роботу в Atmel Studio, роздруки екранів добавити у звіт. Вручну перемножити два беззнакових дробових числа заданого формату.

Таблиця 7 – Варіанти завдань

№	**Завдання 1** Розробити алгоритм і написати програму на асемблері	**Завдання2** Вручну перемножити два дробових числа
01	Додавання двох 4-байтових чисел із знаком	a=1.25, b=0.45, формат 1.7
02	Віднімання двох 4-байтових чисел із знаком	a=1.26, b=3.55, формат 2.6
03	Множення двох цілих чисел без знаку (простий)	a=1.7, b=3.5, формат 3.5
04	Множення двох цілих чисел з додаванням ряду часткових добутків	a=2.5, b=5.3, формат 4.4
05	Множення двох цілих чисел без знаку (починаючи з молодших розрядів)	a=9.1, b=4.5, формат 5.3
06	Множення двох цілих чисел без знаку (mpy8u)	a=23.1, b=3.5, формат 6.2
07	Множення двох цілих чисел із знаком (mpy8s)	a=56.1, b=3, формат 7.1
08	Ділення двох цілих чисел з відновлення залишку	a=1.33, b=0.25, формат 1.7
09	Ділення двох цілих чисел без відновлення залишку	a=1.34, b=8.35, формат 2.6
10	Ділення двох беззнакових чисел (div8u)	a=13.5, b=10.45, формат 3.5
11	Ділення двох знакових чисел (div8s)	a=6.66, b=2.55, формат 4.4
12	Множення двох дробових чисел	13.1, b=2.5, формат 5.3
13	Генерування двох випадкових чисел	a=23.1, b=3.7, формат 6.2
14	Розрахунок CRC8 методом побітового зсуву	a=56.1, b=3, формат 7.1
15	Додавання двох 4-байтових чисел із знаком	a=1.39, b=0.95, формат 1.7
16	Віднімання двох 4-байтових чисел із знаком	a=1.41, b=1.15, формат 2.6
17	Множення двох цілих чисел без знаку (простий)	a=14.2, b=15.33, формат 3.5
18	Алгоритм і програма множення двох цілих чисел з додаванням ряду часткових добутків	a=3.33, b=3.55, формат 4.4
19	Множення двох цілих чисел без знаку (починаючи з молодших розрядів)	a=5.1, b=3.5, формат 5.3
20	Алгоритм і програма множення двох цілих чисел без знаку (mpy8u)	a=15.1, b=4.6, формат 6.2
21	Алгоритм і програма множення двох	a=33.1, b=3

	цілих чисел із знаком (mpy8s)	формат 7.1
22	Ділення двох цілих чисел з відновлення залишку	a=1.33, b=0.25, формат 1.7
23	Ділення двох цілих чисел без відновлення залишку	a=1.48, b=6.85, формат 2.6
24	Ділення двох беззнакових чисел (div8u)	a=1.49, b=0.95, формат 3.5
25	Ділення двох знакових чисел (div8s)	a=2.55, b=5.55, формат 4.4

Примітка. Номер варіанту завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

- завдання до роботи;
- короткий опис теоретичної частини;
- блок схему алгоритму згідно ДСТУ;
- текст програми із коментарями;
- роздрук екранів із вмістом регістрів і пам'яті SRAM після виконання програми у середовищі Microchip Studio.

Питання.

1. Як отримати обернений і доповняльний код від'ємного числа?
2. Які команди асемблера дають обернений і доповняльний код числа?
3. Відняти "на папері" два числа 59 – 24 у доповняльному коді і порівняти з машинним результатом.
4. Пояснити алгоритми додавання знакових і беззнакових 8 і 16-бітових чисел.
5. Пояснити алгоритми множення знакових і беззнакових 8 і 16-бітових чисел.
6. Пояснити алгоритми ділення знакових і беззнакових 8 і 16-бітових чисел.
7. Пояснити алгоритм операції множення з дробовими числами.
8. Пояснити, як перемножити два числа з фіксованою крапкою методом масштабування.
9. Пояснити формати 1.7 і 1.15 для дробових чисел.
10. Пояснити, як перемножити два дробових числа формату 5.3.

ЛАБОРАТОРНА РОБОТА № 4. Арифметичні операції з двійково-десятковими числами

Мета роботи – вивчення способів подання двійково-десяткових чисел в МК AVR.

1. Операції з двійково-десятковими числами

Двійково-десяткові (2-10, BCD) числа зберігаються у двох видах – запакованому і незапакованому. В незапакованому виді на кожну десяткову цифру виділяється байт (цифра займає молодшу тетраду, а старша тетрада залишається нульовою). Таке зберігання цифр є неекономним, тому 2-10 числа зберігаються в регістрах завжди запакованими – молодшу тетраду займає одна цифра, а старшу – інша цифра.

Тому при роботі з 2-10 числами виникають наступні задачі:

- перетворення двійкового 8- або 16-бітового числа в запаковане 2-10;
- розпакування 2-10 у незапакований формат;
- перетворення 2-10 числа у 8- або 16-бітове двійкове число.

Приклади таких перетворень містяться в файлі каталогу прикладів STUDIO /Atmel/Avr tools/Avrassembler/Appnotes/avr204.asm.

Приклад 1.

```
;**** A P P L I C A T I O N   N O T E   A V R 2 0 4 ************************
;* Title:  BCD (2-10) арифметика
;* Двійкове 8-біт у BCD число
;* BCD число двійкове 8-бітове
;* додавання двоцифрових BCD чисел
;* віднімання двоцифрових BCD чисел
;*******************************************************************************
     ; atmega88
     rjmp  RESET ; очищення обробника
;*******************************************************************************
;* "bin2BCD8" - Перетворення 8-бітового бінарного числа у незапаковане BCD число:
;*  8-бітове число (fbin) у 2-цифрове BCD число (tBCDH:tBCDL).
;*  Для отримання запакованого BCD числа замінити вказані рядки
;*******************************************************************************
;***** Регістрові змінні підпрограми
.def  fbin  =r16       ;8-бітове двійкове число
.def  tBCDL =r16       ;старше  (MSD) BCD число
.def  tBCDH =r17       ;молодше (BCD) BCD число

;***** Код
bin2bcd8:
     clr   tBCDH      ;очистити старше (MSD) число результату
bBCD8_1:subi fbin,10       ;input = input - 10
     brcs bBCD8_2          ;перейти на позначку, якщо прапор carry встановлений
     inc tBCDH       ;інкремент старшого числа MSD
;-------------------------------------------------------------------------
; замінити верхній рядок нижнім для запакованого BCD виведення
;     subi  tBCDH,-$10      ;tBCDH = tBCDH + 10
;-------------------------------------------------------------------------
     rjmp bBCD8_1          ; повторення циклу
bBCD8_2:subi fbin,-10 ; компенсація лишнього віднімання
;-------------------------------------------------------------------------
; добавити нижній рядок для запакованого BCD виведення
;     add   fbin,tBCDH
;-------------------------------------------------------------------------
     ret

;*******************************************************************************
```

```
;* "BCD2bin8" - перетворення незапакованого BCD числа у 8-бітове бінарне
;  2-цифрове BCD число (fBCDH:fBCDL) у 8-бітове число (tbin).
;* Щоб підпрограма сприймала запаковані BCD числа розкоментувати вказані
;  рядки
;* У такому випадку fBCDH буде завантажуватися BCD числом
;* для перетворення до виклику підпрограми.
;***************************************************************************
;***** Регістрові змінні підпрограми
.def  tbin  =r16       ;бінарний результат
.def  fBCDL =r16       ;молодша цифра BCD вводу
.def  fBCDH =r17       ;старша цифра  BCD вводу

;***** Код
BCD2bin8:
;---------------------------------------------------------------------------
;  Для введення запакованих BCD чисел розкоментувати два наступні рядки
;     mov   tbin,fBCDH ;копіювання введення в результат
;           andi  tbin,$0f   ;очищення старшої тетради результату
;---------------------------------------------------------------------------

BCDb8_0:subi    fBCDH,1    ;fBCDH = fBCDH - 1
      brcs  BCDb8_1    ;перейти, якщо прапор carry встановлений
;---------------------------------------------------------------------------
; Для введення запакованих BCD чисел замінити два верхні рядки
; двома закоментованими рядками
;     subi  fBCDH,$10  ;MSD = MSD - 1
;     brmi  BCDb8_1    ;перейти, якщо прапор Neg встановлений
;---------------------------------------------------------------------------
      subi  tbin,-10   ;відсутня команда adi,result = result + 10
      rjmp  BCDb8_0    ;повторити цикл
BCDb8_1:ret                ;інакше return

;***************************************************************************
;* "BCDadd" - додавання 2-розрядних запакованих BCD чисел
;* додавання двоцифрових BCD чисел BCD1 і BCD2.
;* Результат повертається в BCD1, а переповнення прапора carry в BCD2.
;***************************************************************************
;***** Регістрові змінні підпрограми
.def  BCD1=r16         ;вхідне значення 1 числа BCD
.def  BCD2=r17         ;вхідне значення 2 числа BCD
.def  tmpadd=r18       ;регістр тимчасових значень

;***** Код
BCDadd:
      ldi   tmpadd,6   ;значення для додавання
      add   BCD1,BCD2  ;двійкове додавання чисел
      clr   BCD2       ;очищення прапора carry для BCD2 числа
      brcc  add_0 ;якщо прапор очищено
      ldi   BCD2,1     ;встановити BCD carry
add_0:      brhs  add_1 ;перейти, якщо прапор half carry встановлено
      add   BCD1,tmpadd      ;додати 6 to молодшої тетради (LSD)
      brhs  add_2      ;якщо прапор half carry встановлено (LSD <= 9)
      subi  BCD1,6     ;відновити значення
      rjmp  add_2      ;інакше
add_1:      add   BCD1,tmpadd      ;додати 6 до LSD
add_2:      swap  tmpadd
      add   BCD1,tmpadd      ;додати 6 до старшої тетради (MSD)
      brcs  add_4      ;якщо прапор carry встановлено (MSD <= 9)
      sbrs  BCD2,0           ;пропустити якщо біт 0 у регістрі BCD2 встановлено
      subi  BCD1,$60   ;відновити значення
add_3:      ret         ;інакше
add_4:      ldi   BCD2,1;    встановити прапор BCD carry
      ret

;***************************************************************************
```

```
;* "BCDsub" - віднімання 2-розрядних запакованих BCD чисел
;* віднімання дворозрядних беззнакових BCD чисел BCDa - BCDb.
;* Результат повертається в BCDa і прапор переповнення в BCDb.
;***************************************************************************
;***** Регістрові змінні підпрограми
.def  BCDa  =r16  ;вхідне значення першого  BCD числа
.def  BCDb  =r17  ;вхідне значення другого BCD числа

;***** Код
BCDsub:
     sub   BCDa,BCDb  ;віднімання бінарних чисел
     clr   BCDb
     brcc  sub_0      ;якщо прапор carry очищений
     ldi   BCDb,1     ;зберегти прапор carry у BCDB1, bit 0
sub_0:    brhc  sub_1 ;якщо half carry очищений
     subi  BCDa,$06   ;    LSD = LSD - 6
sub_1:    sbrs  BCDb,0 ;пропустити, якщо біт 0 в рег. BCDb встановлений
     ret              ;return
     subi  BCDa,$60   ;відняти 6 від MSD
     ldi   BCDb,1     ;встановити прапор underflow carry
     brcc  sub_2      ;якщо прапор carry очищено
     ldi   BCDb,1     ;очистити прапор underflow carry
sub_2:    ret

;***** Регістрові змінні основної програми
.def  temp  =r16  ; регістр для тимчасових змінних
;***** Code

RESET:
     ldi   temp,low(RAMEND)
     out   SPL,temp
     ldi   temp,high(RAMEND)
     out   SPH,temp   ;ініціалізація стеку

;***** Перетворення 55 у 2-байтове BCD

     ldi   fbin,55
     rcall bin2BCD8   ;результат: tBCDH:tBCDL = 0505
/*
;***** Перетворення $0403 (43) у 8-бітове бінарне число
     ldi   fBCDL,3
     ldi   fBCDH,4
     rcall BCD2bin8   ;результат: tbin = $2b (43)

;***** Додавання BCD чисел 51 і 79
     ldi   BCD1,$51
     ldi   BCD2,$79
     rcall BCDadd     ;результат: BCD2:BCD1=$0130

;***** Віднімання BCD чисел 72 - 28
     ldi   BCDa,$72
     ldi   BCDb,$28
     rcall BCDsub     ;результат: BCDb=$00 (позитивний результат), BCDa=44
*/
```

2. Додавання двійково-десяткових чисел

При додаванні двох двійково-десяткових чисел $A=a_{n-1}a_{n-2}...a_1a_0$ і $B=b_{n-1}b_{n-2}...b_1b_0$ поступають наступним чином. Якщо обидва операнди мають однакові знаки, то виконують додавання модулів цих чисел (|A|+|B|), а знаковий розряд суми визначають по знаку одного з доданків. Якщо операнди мають різні знаки, то попередньо знак суми встановлюють по знаку першого операнда А. Потім виконують віднімання модулів чисел (|A|-|B|). Якщо отримана

різниця більша нуля, то знак суми зберігається без змін. Якщо різниця менша нуля, то потрібно знайти доповняльний код різниці і змінити знак суми на протилежний.

При додаванні двох чисел А і В, у 2-10 коді, з вагою розрядів 8-4-2-1 в кожній тетраді, в одному розряді суми S = A + B можна отримати результати:

1) $s_i \le 9$ - не потрібне корегування;

2) $10 \le s_i \le 15$ - потрібне корегування збільшення s_i на 6 зі створенням перенесення з тетради;

3) $s_i > 15$ – потрібне корегування збільшення s_i на 6. В цьому випадку перенесення з тетради утворюється автоматично при додаванні операндів до виконання корекції.

На практиці корегування виконують наступним чином. При додаванні 2-10 чисел до суми додають число, у якого кожний десятковий розряд 6. У цьому випадку, якщо обчислювана порозрядна сума $s_i \le 9$, то перенесення з тетради не виникає і надлишкове значення шість підлягає вилученню. У всіх інших випадках добавлене в розряд значення шість вилучається автоматично з перенесенням з тетради у процесі додавання. При такому способі додавання програмна реалізація спрощується, так як для корегування результату в тетраді перевіряється лише один признак – наявність або відсутність перенесення з тетради.

Таким чином, додавання чисел без знаку (як і модулів |A| + |B|) можна виконати за алгоритмом, схема якого зображена на рис. 1:

1) двійково-десятковий код першого операнда $a_{n-1}a_{n-2}...a_1a_0$ додається до коду 66...66, утворюючи першу проміжкову суму $s'_{n-1}s'_{n-2}...s'_1s'_0$;

2) до отриманої суми додають двійково-десятковий код другого операнду $b_{n-1}b_{n-2}...b_1b_0$, утворюючи другу проміжкову суму $s''_{n-1}s''_{n-2}...s''_1s''_0$;

3) потетрадно корегується результат. Правило корегування формулюється наступним чином: якщо в результаті другого додавання перенесення з i-тетради відсутнє ($c_{i+1}=0$), то з s'' віднімається шість (або додається $10 = [-6]_{доп}$). При виникненні перенесення з i-тетради корегування не виконується, а отриманий результат s'' є істинним. При корегуванні перенесення з тетради не повинно змінити вміст наступної тетради суми s''_{i+1}.

При побайтовій обробці довгих операндів у 8-розрядному процесорі для запобігання помилок корегування необхідно виконувати окремо для кожної тетради.

Приклад. A = 50, B = 25. Знайти суму A + B. Двійково-десяткове подання чисел A і B: A = 0101 0000, B = 0010 0101. Додаючи числа 66+A+B, отримаємо

```
  0110 0110    66
+ 0101 0000    50
  1011 0110

  1011 0110
+ 0010 0101    25
  1101 1011
```

Корегуємо молодшу тетраду:

```
  1101 1011
+ 1111 1010    -$06 = $FA
  1101 0101
```

Корегуємо старшу тетраду:

```
  1101 0101
+ 1010 0000    -$60 = $A0
  0111 0101     75
```

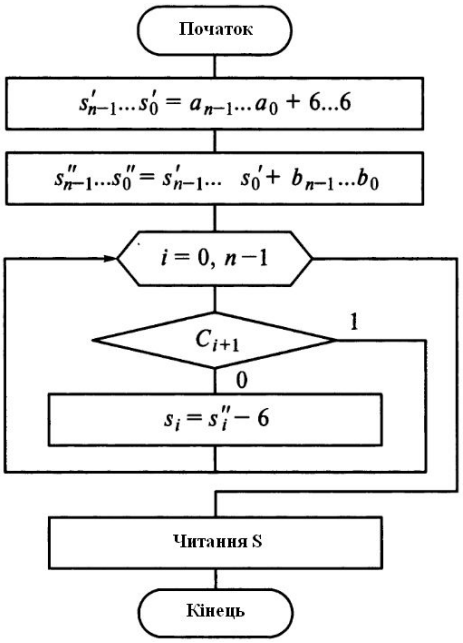


Рисунок 1 – Блок схема додавання беззнакових 2-10 чисел

3. Віднімання двійково-десяткових чисел

Віднімання беззнакових чисел A-B можна виконати за алгоритмом, схема якого зображена на рис. 2:

1) виконати віднімання A-B (додаючи A до доповняльного коду B), утворюючи перший проміжковий результат R'. Якщо в результаті операції виникає перенесення з старшої тетради (при цьому прапор позики дорівнює 0), результат є позитивним. При відсутності перенесення (прапор позики дорівнює 1) результат є негативним і його потрібно перевести в доповняльний код R".

2) позитивний результат корегується за наступним правилом, сформульованим для додавання 2-10 чисел. Корегування негативного результату виконується інакше: якщо відбулося перенесення з *i*-тетради (прапор міжтетрадної позики дорівнює 0) при відніманні A-B, то з і-ої тетради R" віднімається шість. Корегування перенесення з тетради не повинно міняти вміст наступної тетради проміжкового результату.

Приклад. A = 23, B = 62. Знайти різницю A-B. Двійково-десяткове подання чисел A і B: A = 0010 0011, B = 0110 0010.

Доповнення числа B: $[B]_{доп}$ = 1001 1110. Віднімаючи A - B = A + $[B]_{доп}$, отримаємо R':

```
  0010 0011
+ 1001 1110
  1100 0001
```

Перенесення немає: результат негативний, прапор позики дорівнює 1.

Формується доповнення $[R']_{доп}$ і виконується корегування:

```
  0011 1111
+ 1111 1010      -$06 = $FA
  1111 1001
```

Результат: прапор знаку -, різниця 39.

Коли негативний результат потрібно зберегти в доповняльному коді, то крок формування доповнення $[R']_{доп}$ пропускається. Можна зразу переходити до корегування (віднімання шість в тих тетрадах, де не було перенесення).

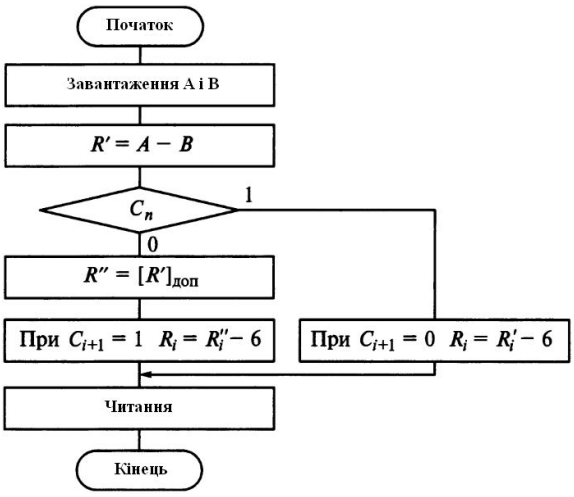


Рисунок 2 – Блок схема віднімання беззнакових 2-10 чисел

Програма додавання і віднімання двійкових і двійково-десяткових однобайтних чисел. Додавання/віднімання двійкових чисел можна виконувати із знаком або без знаку, а двійково-десяткових чисел – у запакованому форматі без знаку.

Приклад 2.

```
;
;*********************************************************************
;Програма додавання і віднімання однобайтних операндів
;*********************************************************************
; atmega88"
.include "m88def.inc"
.def  BCDa   =r30
.def  BCDb   =r31
.def  tmpadd =r29
.def  temp   =r28
.org 0

      rjmp  INIT
;*********************************************************************
;Підпрограма додавання 2-10 упакованих беззнакових чисел
;BCDa и BCDb. Результат повертається в BCDa, перенесення в BCDb.
;*********************************************************************
BCDadd:
      ldi   tmpadd,$66
      add   BCDa,tmpadd  ;  $51+$66=$b7
      add   BCDa,BCDb    ;  $b7+$79=$30, C=1,H=1
      clr   BCDb
      brcs  add_0  ; перейти, якщо прапор carry встановлений
      rjmp  add_1
add_0:     ldi   BCDb,1     ;Встановити вихідне перенесення  0x01
add_1:     brhs  add_2 ;перейти, якщо прапор hulf встановлений
      subi  BCDa,$06   ; LSD = LSD - 6
add_2:     sbrs  BCDb,0     ;пропустити, якщо біт 0 регістра BCDb дорівнює 0,
      subi  BCDa,$60   ; MSD = MSD - 6
      ret                ;  BCDb:BCDa = 0x01:0x30,  51+49=130

;*********************************************************************
```

```
;Підпрограма віднімання 2-10 запакованих беззнакових чисел
;BCDa и BCDb (BCDa - BCDb).
;Результат повертається в BCDa,знак різниці в BCDb.
;*****************************************************************
BCDsub:
      sub   BCDa,BCDb
      clr   BCDb
      brcc  sub_0   ;перейти, якщо прапор carry очищений
      ldi   BCDb,1        ;  зберегти його
sub_0:      brhc  sub_1   ; перейти, якщо прапор hulf очищений
      subi  BCDa,$06 ; LSD = LSD - 6
sub_1:      sbrs  BCDb,0   ; пропустити, якщо біт 0 регістра BCDb встановлений
      ret          ; вийти
      subi  BCDa,$60 ;інакше відняти $60
      ret

;*****************************************************************
;Основна програма
;*****************************************************************
INIT:
      ldi   temp,low(RAMEND)
      out   SPL,temp
      ldi   temp,high(RAMEND)
      out   SPH,temp
;*** Add BIN
loop: ldi   BCDa,51
      ldi   BCDb,-79
      add   BCDa,BCDb  ;Результат двійковий: BCDa=$E4
;*** Sub BIN
      ldi   BCDa,72
      ldi   BCDb,28
      sub   BCDa,BCDb  ;Результат двійковий: BCDa=$2C
;*** Add BCD Unsigned
      ldi   BCDa,$51
      ldi   BCDb,$79
      rcall BCDadd     ;Результат 2-10: BCDb:BCDa=$0130
;*** Sub BCD Unsigned
      ldi   BCDa,$72
      ldi   BCDb,$28
      rcall BCDsub     ;Результат 2-10: BCDb=$00-позитивний,BCDa=44
      ldi   BCDa,$00
      ldi   BCDb,$90
      rcall BCDsub     ;Результат 2-10: BCDb=$01-негативний,BCDa=10
            rjmp  loop
```

Завдання.

№	Зміст завдання
1, 15	Написати програму на асемблері, яка перетворює 8-бітове бінарне число у незапаковане BCD число.
2, 16	Написати програму на асемблері, яка перетворює 16-бітове бінарне число у незапаковане BCD число.
3, 17	Написати програму на асемблері, яка перетворює 8-бітове бінарне число у запаковане BCD число.
4, 18	Написати програму на асемблері, яка перетворює 16-бітове бінарне число у запаковане BCD число.
5, 19	Написати програму на асемблері, яка перетворює 8-бітове незапаковане BCD число у бінарне число.

6, 20	Написати програму на асемблері, яка перетворює 16-бітове незапаковане BCD число у бінарне число
7, 21	Написати програму на асемблері, яка перетворює 8-бітове бінарне число у запаковане BCD число.
8, 22	Написати програму на асемблері, яка перетворює 16-бітове бінарне число у запаковане BCD число.
9, 23	Написати програму на асемблері, яка додає 8-бітові беззнакові запаковані BCD числа.
10, 24	Написати програму на асемблері, яка додає 16-бітові беззнакові запаковані BCD числа.
11, 25	Написати програму на асемблері, яка віднімає 8-бітові беззнакові запаковані BCD числа (перший операнд більший другого).
12, 25	Написати програму на асемблері, яка віднімає 16-бітові беззнакові запаковані BCD числа (перший операнд більший другого).
13	Написати програму на асемблері, яка віднімає 8-бітові беззнакові запаковані BCD числа (перший операнд менший другого).
14	Написати програму на асемблері, яка віднімає 16-бітові беззнакові запаковані BCD числа (перший операнд менший другого).

Примітка. Номер варіанту завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

- завдання до роботи;
- короткий опис теоретичної частини;
- блок схему алгоритму згідно ДСТУ;
- текст програми із коментарями;
- роздрук екранів із вмістом регістрів і пам’яті SRAM після виконання програми у середовищі Microchip Studio.

Питання.

1. Які числа називаються 2-10 десятковими і які є формати їх зберігання.
2. Розказати про алгоритм перетворення двійкових 8-бітових чисел у 2-10 числа.
3. Розказати про алгоритм перетворення 2-10 чисел 8-бітові двійкові.
4. Розказати про алгоритм додавання 2-10 чисел.
5. Розказати про алгоритм віднімання 2-10 чисел.

ЛАБОРАТОРНА РОБОТА № 5. Порти введення-виведення. Світлодіоди, кнопки

Мета роботи – вивчення плати STK500, засобів розробки Microchip Studio/Atmel Studio/AVR Studio, програмування портів введення-виведення, підключення світлодіодів, практична робота з симулятором Proteus.

1. Практична частина

1.1. Плата STK500 фірми ATMEL

STK500 є універсальною платою розробника, яка дозволяє створювати застосування разом з інтегрованим середовищем проектування AVR Studio 4, Microchip Studio. Плата STK500 підтримує цілий ряд МК, в тому числі і модель ATmega8515, для чого служать відповідні панелі для монтування і засоби комунікації, рис. 1.

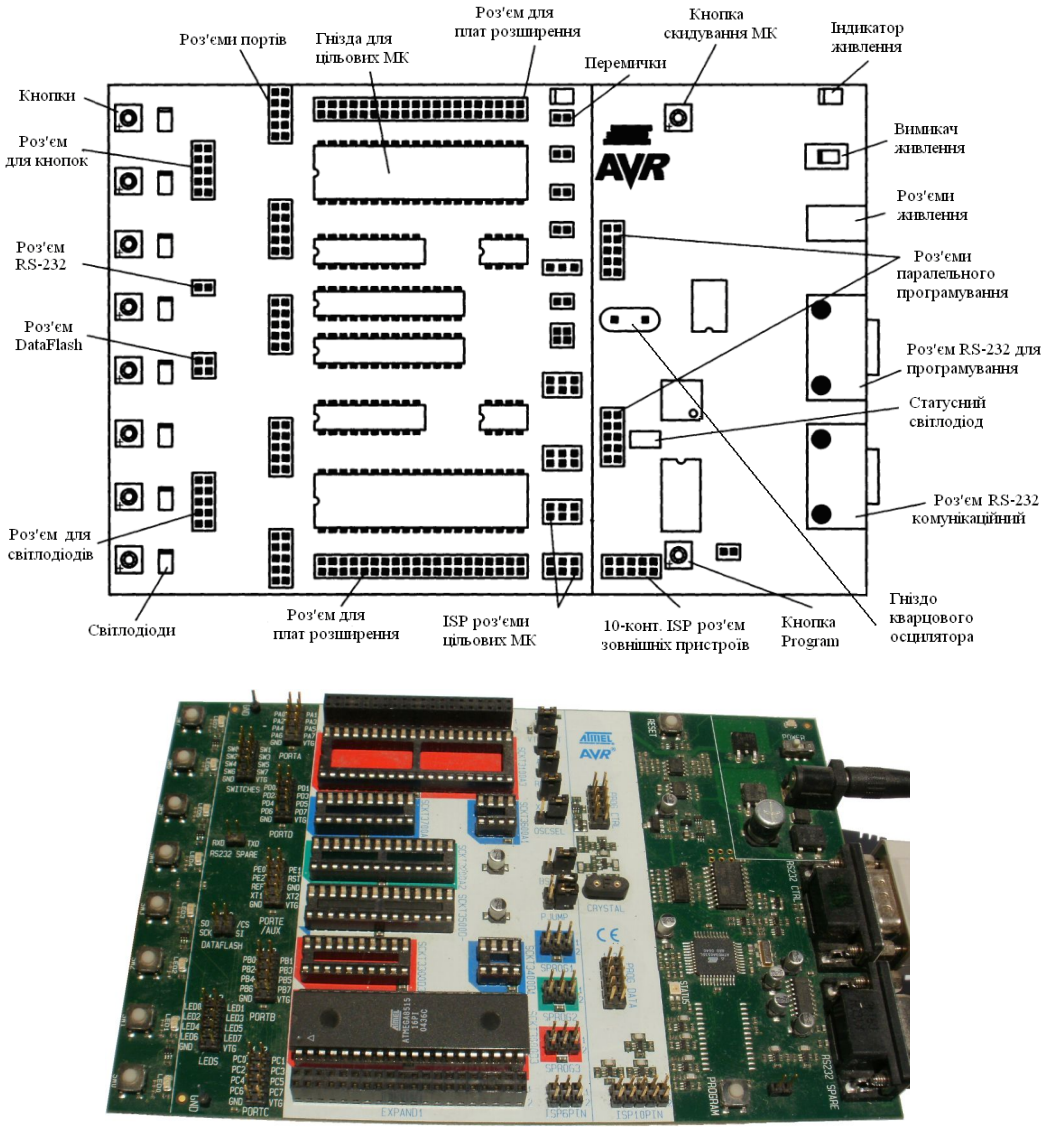


Рисунок 1 – Вигляд плати STK500 і розміщення її засобів комутації

Початкові встановлення перемичок забезпечують роботу МК сумісно з тактовим генератором і стабілізатором напруги, встановленим на платі STK500. Набір також має засоби введення і індикації, інтерфейс RS-232, засоби розширення для підключення зовнішніх пристроїв.

Плата STK500 має:

- стабілізоване джерело живлення з вхідною напругою 10…15В і програмно керованою вихідною напругою;
- вісім кнопок загального призначення;
- вісім світлодіодів загального призначення;
- роз'єми всіх портів введення/виводу МК;
- 8-, 20-, 28-, 40-вивідні панелі для встановлення DIP-корпусів МК AVR;
- інтерфейс RS-232 для програмування і керування з програми AVR Studio 4, встановленої на персональному комп'ютері;
- додатковий порт RS-232 загального призначення;
- роз'єми розширення для підключення зовнішніх модулів при макетуванні;
- пам'ять DataFlash ємністю 2 Мбіт для енергонезалежного зберігання даних;
- засоби підтримки паралельного і послідовного програмування підвищеною напругою усіх МК AVR;
- засоби послідовного внутрішньо-системного програмування (ISP) усіх МК AVR;
- внутрішньо-системний програматор для програмування МК безпосередньо в цільовому застосуванні.

Світлодіоди і кнопки загального призначення. Плата STK500 має вісім жовтих світлодіодів і вісім кнопок без фіксації. Світлодіоди і кнопки електрично відділені від решти плати і підключені до власних роз'ємів. Вони можуть бути підключені до МК AVR 10-провідними шлейфами через роз'єми портів введення/виведення. Схема підключення кнопки показана на рис. 2. При натисканні кнопки на виводі SWn встановлюється низький рівень напруги (GND), а при відпусканні – високий (VTG). Робочий діапазон напруг VTG =1,8.. .6,0 В.

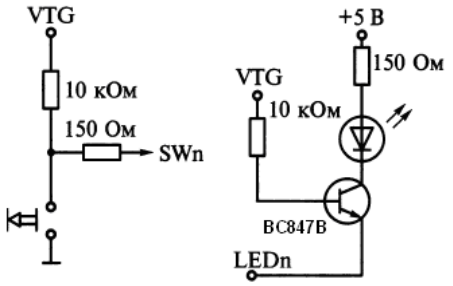
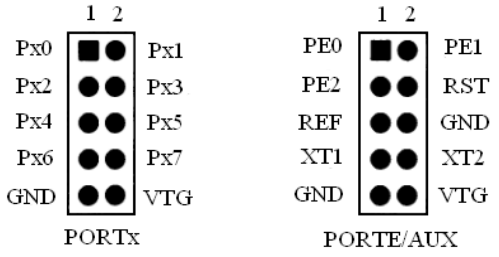


Рисунок 2 – Схема включення кнопки і світлодіода

Виводи світлодіодів LEDx і кнопок SWx (x = 0...7) з'єднані з відповідними контактами роз'ємів SWITCHS і LEDS. Контакти 9 і 10 роз'ємів використані для сигналів GND і VTG. Тому необхідно з'єднувати шлейфами однойменні виводи роз'ємів індикаторів і кнопок з портами МК (шлейф не повинен перекручуватися). Для цього в шлейфі червоним кольором виділена одна з ліній, яка повинна з'єднувати однойменні виводи роз'ємів (наприклад, LED0 и Px0). Особливості роботи світлодіодів і кнопок необхідно враховувати при програмуванні портових операцій МК, зв'язаних із зверненням до світлодіодів і кнопок.

Роз'єми портів. Любий порт введення/виведення МК AVR може бути підключений до світлодіодів і кнопок за допомогою 10-провідного шлейфу. На роз'єми в додаток до ліній портів виведені напруга живлення цільового МК VTG (VCC) і загальний провід GND.

Розміщення виводів роз'ємів і їх відповідність лініям портів введення/виведення зображено на рис. 3. Роз'єм порту E (PORTE/AUX) має спеціальні сигнали і функції в доповнення до ліній порту PE.



а *б*

Рисунок 3 – Виводи роз'ємів портів МК: *а*) Px (x=A,B,C,D); *б*) PE

Інтерфейс RS-232 для користувача. Плата STK500 має два порти RS-232 CTRL, RS232 SPARE. RS232 CTRL використовується для зв'язку з AVR Studio або Microchip Studio. RS232 SPARE використовується для зв'язку МК, встановленого на платі, з комп'ютером через його послідовний порт RS-232 (COM-порт). Для цього два виводи каналу UART МК потрібно фізично з'єднати з входами порту RS-232, виведеними на 2-штифтовий роз'єм RS232 SPARE. Порт RS-232 на платі STK містить схему перетворення рівнів сигналів інтерфейсу.

Flash-пам'ять даних DataFlash. Плата STK500 має мікросхему Flash-пам'яті AT45D021 ємністю 2 Мбіт родини DataFlash, яка може бути використана для енергонезалежного зберігання даних. DataFlash – Flash-пам'ять з послідовним програмуванням через SPI-інтерфейс, може бути підключена до ліній порту PB МК. Для цього необхідно використати 4-штифтовий роз'єм з позначкою DATAFLASH, який зв'язаний з SPI-інтерфейсом DataFlash. Для з'єднання роз'єму DATAFLASH з лініями порту PB необхідно з'єднати PB6-SO, PB7-SCK, PB4-/CS, PB5-SI.

Секція цільових панелей. Модуль програмування складається з восьми панелей в центрі плати. В одну з них необхідно встановити цільовий МК AVR для програмування і подальшого використання в застосуваннях. Для Flash-пам'яті МК AVR гарантована зносостійкість складає 1000 циклів програмування.

Секція перемичок і програмування. Керуючий МК і вісім перемичок визначають роботу плати STK-500. В поставці ці перемички встановлені в початкове положення.

Після вставлення МК в панель можна його програмувати, для чого використовується AVR Studio/Microchip Studio і один з можливих методів:

- внутрішньо-системне програмування при нормальній напрузі живлення. Це основний метод при виконанні лабораторних робіт;

- програмування підвищеною напругою, при якій напруга живлення завжди дорівнює 5 В. Допускається підключення ланцюгів VTARGET, RESET, XTAL1 і AREF до секції панелей.

Інші апаратні компоненти. Плата STK500 має два роз'єми розширення, встановлені з двох сторін від секції цільових панелей. Всі сигнали портів введення/виведення МК AVR, сигнали програмування і керування є на виводах цих роз'ємів. Роз'єми розширення дозволяють підключати макети застосувань до STK500.

STK500 має дві кнопки спеціального призначення і три світлодіоди для індикації стану. Натискання кнопки RESET скидає цільовий МК. Нові версії AVR Studio/Microchip Studio здатні

оновити програму керуючого МК. Для цього користувачу потрібно натиснути на кнопку PROGRAM після подачі напруги живлення на STK500. Основний індикатор напруги живлення – червоний світлодіод, безпосередньо підключений до основного джерела живлення STK500. Даний індикатор має безперервно світитися після включення живлення на STK500 перемикачем POWER. Індикатор цільової напруги – світлодіод, зв'язаний з лінією живлення VCC (VTG) цільового МК. Індикатор безперервно світиться при наявності напруги живлення на цільових панелях МК. Статусний світлодіод трикольоровий. При програмуванні він жовтий, після успішного завершення програмування стає зеленим. Червоний колір показує, що програмування було перервано. При програмуванні статусний світлодіод послідовно змінює свій стан від червоного через жовтий до зеленого для індикації готовності цільового МК.

1.2. Інтерфейс STK500 в AVR Studio

Як програмне застосування для зв'язку з платою STK500 використовується AVR Studio/Microchip Studio/Atmel Studio 7 . Плата під'єднується до ПК через COM-порт . Так як сучасні ПК не мають COM-порту, то використовується перехідник Serial-USB між STK500 і USB портом ПК. До перехідника Serial-USB для ПК з Windows потрібно встановити драйвер CP340 (Win XP) або CH340 (Win 10).

Вибір з меню AVR Studio команди /Tools/Program AVR/Connect відкриває вікно вибору плати і номеру COM-порту, рис. 4. Після натискання кнопки Connect, у випадку успішного розпізнання плати, появляється вікно інтерфейсу користувача плати STK500, рис. 5.

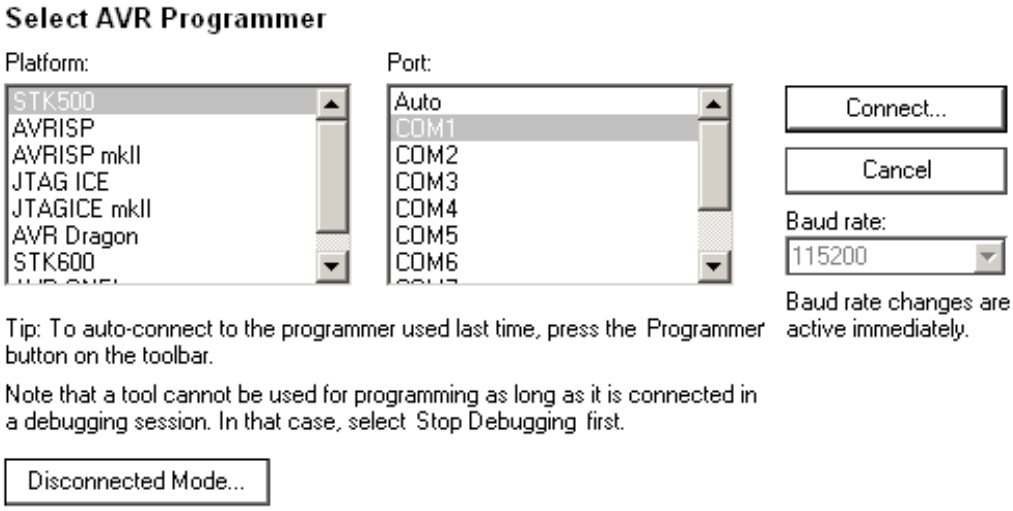


Рисунок 4 – Вибір плати і номеру COM-порту

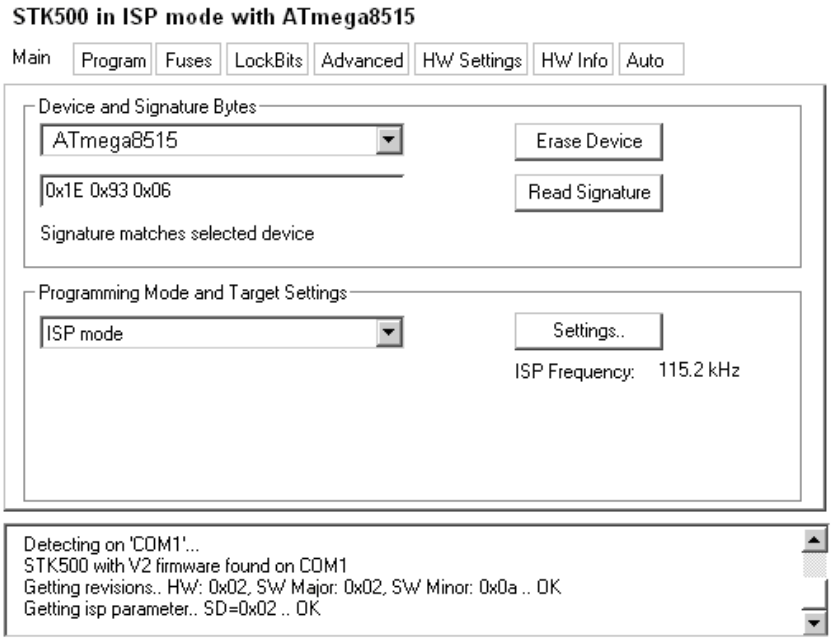


Рисунок 5 – Вікно Main інтерфейсу користувача

Вікно *Main* дозволяє вибрати тип цільового МК, очистити пам’ять МК (FLASH, EEPROM) – кнопка Erase Device, прочитати сигнатурні байти (ідентифікації МК і його виробника) – кнопка Read Signature, вибрати режим програмування МК (ISP – In-System programming), встановити тактову частоту при програмування – кнопка Settings.

Вікно *Program* розділене на п’ять областей Device, Flash, EEPROM, Input HEX file, Input ELF file, поле історії, рис. 6. Кнопка Erase Device (в області Device) і прапорець *Erase device before flash programming* активізує функцію стирання пам’яті перед програмуванням, а встановлення прапорця *Verify device after programming* дозволяє перевіряти правильність записаної інформації у FLASH і EEPROM пам’ять.

В області програмування FLASH пам’яті вказується шлях до вхідного hex-файлу. Кнопки Program, Verify, Read активують функції запису, перевірки і читання FLASH пам’яті.

В області програмування EEPROM пам’яті вказується шлях до вхідного hex-файлу. Кнопки Program, Verify, Read активують функції запису, перевірки і читання EEPROM пам’яті.

В області програмування/збереження ELF файлу вказується шлях до вхідного ELF-файлу. Кнопки Program і Save активують функції програмування і збереження ELF-файлу. ELF-файл (Executable and Linkable Format) може містити дані для FLASH і EEPROM пам’яті, а також конфігурацію бітів fuse і lock в одному файлі.

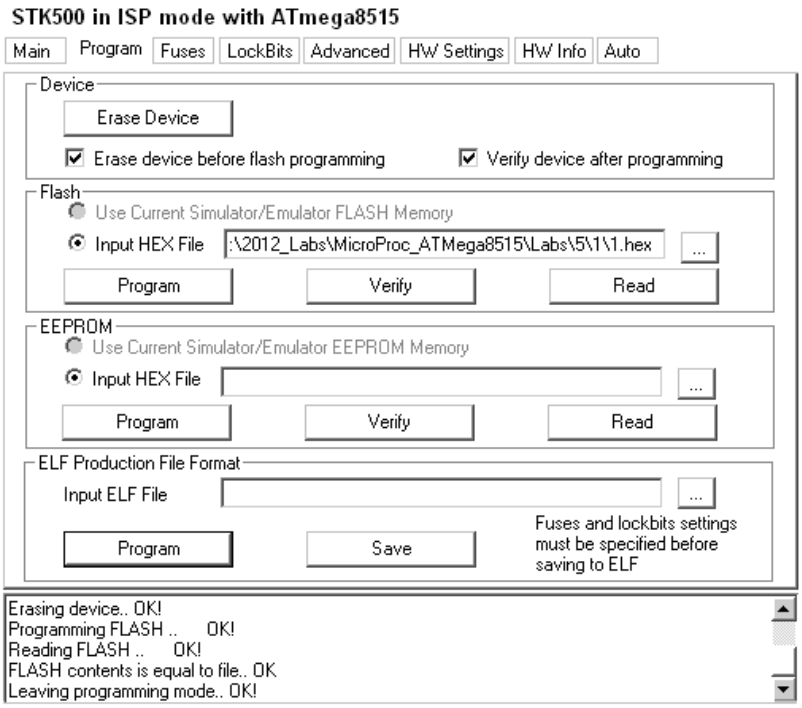


Рисунок 6 – Вікно Program інтерфейсу користувача

Вікно *Fuses* дозволяє задати конфігураційні біти цільового МК, рис. 7. Інформація про доступність бітів в різних режимах програмування і їх описання знаходиться в документації відповідного МК.

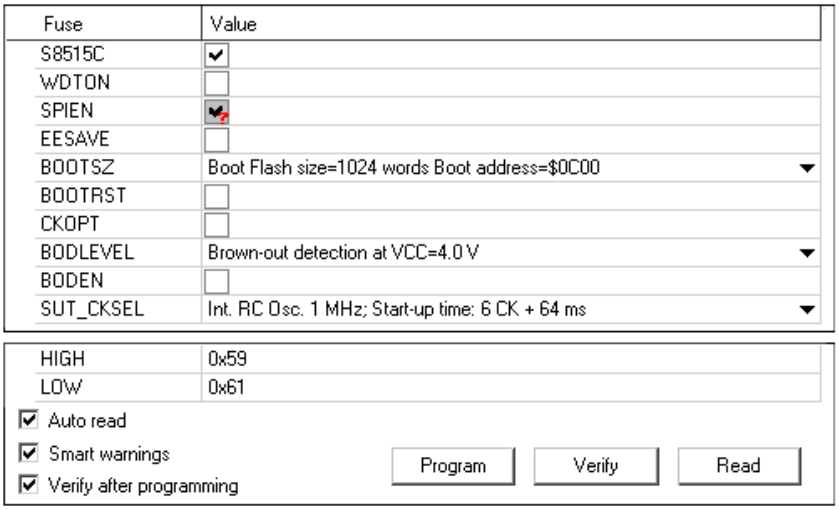


Рисунок 7 – Конфігураційні біти цільового МК ATmega8515

Вікно *LockBits* дозволяє задати біти захисту програми для цільового МК, рис. 8. Режим захисту задається комбінацією бітів. Всі біти захисту доступні як в режимі ISP-програмування, так і в режимі програмування підвищеною напругою.

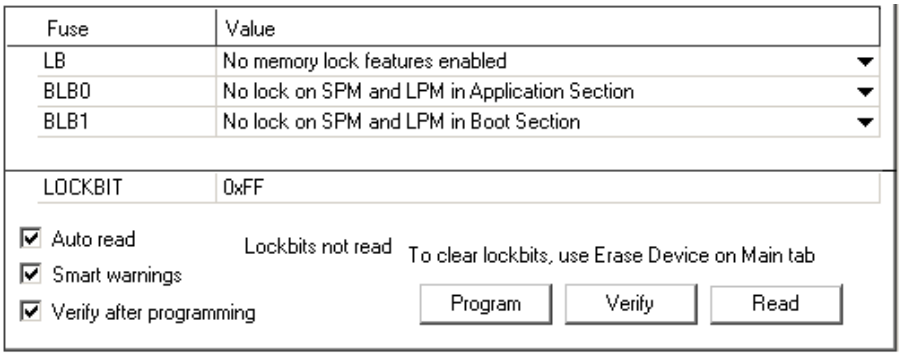


Рисунок 8 – Біти захисту програми цільового МК ATmega8515

Вікно *HW settings* дозволяє задати робочі напруги (Vtarget, Vref) і тактову частоту плати (Clock Generator).

1.3. Інтерфейс STK500 в Atmel Studio 7

Створити в Atmel Studio/Microchip Studio/Atmel Studio 7 новий проект на мові асемблера. Набрати у редакторі код програми. Вибрати пункти меню Project/Assembler Application properties. У вікні, що відкриється у підпункті Build із випадаючого списку вибрати Release, у підпункті Device задати тип МК Atmega 8515, у підпункті Tool з випадаючих списків вибрати тип програматора STK500 і тип інтерфейсу ISP. Вибором пункту Buid/Build assembler application асемблювати програму і отримати hex файл.

Вставити у відповідну панель на платі STK500 МК (в даному випадку Atmega 8515). З'єднати 6-провідниковим шлейфом штифти ISP6PIN з штифтами SPROG3 (щоб штифт 1 був з'єднаний з штифтом 1). В Atmel Studio 7 вибрати пункти меню Tools/Add target. У вікні, що появиться вибрати із випадаючих списків тип програматора і порт, натиснути кнопку Apply.

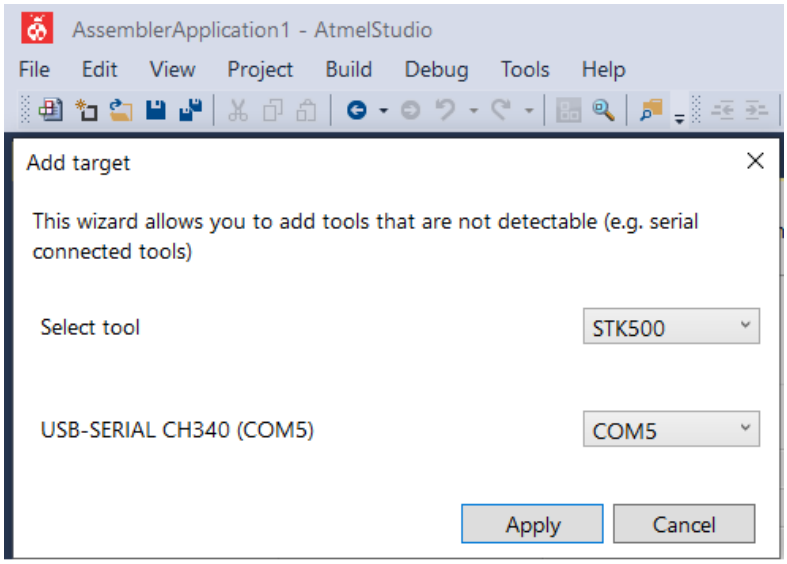


Рисунок 9 – Підключення програматора STK500 в Atmel Studio

Вибрати пункт меню Tools/Device Programming і задати параметри програмування і натиснути кнопку Program. Hex файл, отриманий після компіляції програми, із вибраного каталогу буде записаний у МК:

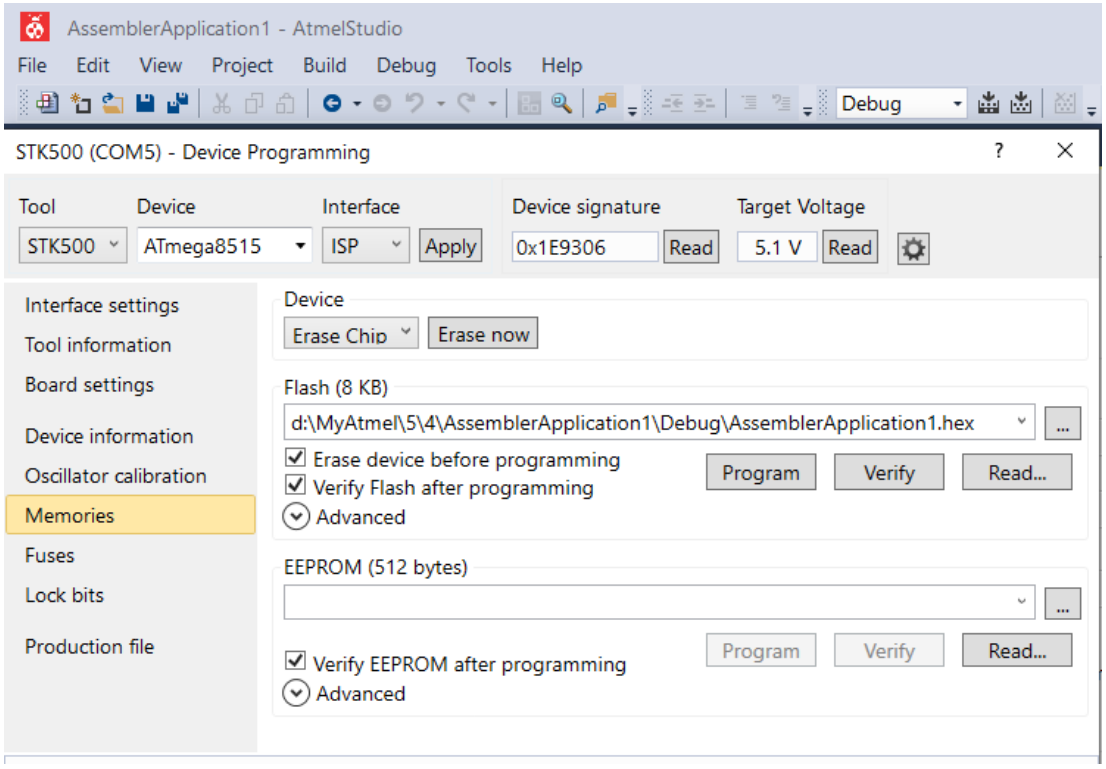


Рисунок 10 – Запис hex-файлу в МК в Atmel Studio/Microchip Studio/Atmel Studio 7

Якщо програма, наприклад, блимає світлодіодами то потрібно відключити живлення, з'єднати 10-провідниковим шлейфом відповідний порт і LEDS штифти. Ввімкнути живлення МК, при цьому програма в МК почне працювати і діоди заблимають у відповідності до заданого алгоритму.

1.4. Паралельні порти введення/виведення

МК ATmega8515 має чотири паралельних 8-розрядних порти Px (x=A, B, C, D) і один 3-х розрядний порт PE. Всі лінії портів можуть програмуватися на введення або виведення даних незалежно один від одного і підключатися через внутрішні підтягуючі резистори з опором 35… 120 КОм до шини живлення Vcc.

В склад кожного порту входять три регістри з іменами DDRx, PORTx, PINx. В МК ATmega8515 регістр PINx не має апаратної реалізації. Це ім'я використовується для читання ліній інтерфейсу. На рис. 11 показана загальна структурна схема 8-розрядних портів Px і структурна схема одного розряду порту Px.Y (Y=0,…,7) МК ATmega8515.

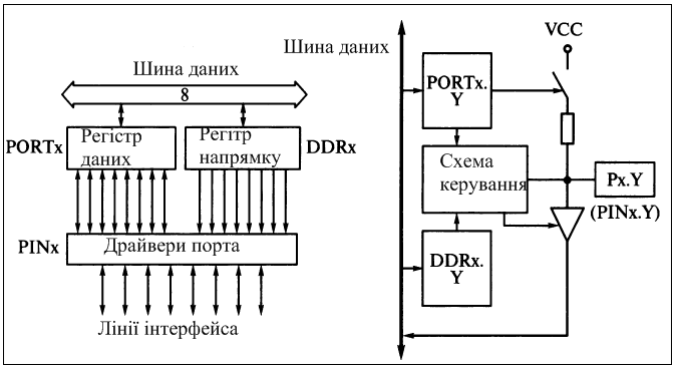


Рисунок 11 – Структура порту Px і схема одного розряду

Стан розряду DDRx.Y визначає напрямок передачі біта даних через вивід порту Px.Y. При DDRx.Y=0 вивід порту Px.Y *є входом* (по замовчуванню), а при DDRx.Y=1 – *виходом*.

Регістр даних PORTx.Y фактично є просто вихідним буфером, все що в нього записується, тут же появляється на виході.

В *режимі входу* стан розряду PORTx.Y визначає стан виведення Px.Y. При PORT.xY=0 внутрішній резистор відключається і вивід Px.Y знаходиться у третьому стані з високим імпедансом (Z-стан). При PORTx.Y=1 вивід порту через внутрішній "підтягуючий" резистор, підключається до шини живлення Vcc. При читанні значень на вході виводів мікросхеми потрібно звертатися до Px.y (PINx).

В *режимі виходу* стан розряду PORTx.Y визначає стан виведення Px.Y. При PORTx.Y=0 на виводі Px.Y встановлюється напруга низького рівня , при PORTx.Y=1 – високого.

При пуску і перезапуску МК всі розряди регістрів DDRx і PORTx скидаються в нульовий стан, внаслідок чого виводи портів працюють в режимі входу і знаходяться в Z-стані.

При сумісному використанні всіх розрядів порту для *введення* байта даних використовуються команди з мнемонікою IN Rd, PINx, а для *виведення* – OUT PORTx, Rr (де d, r =0,…,31). Значення вихідного сигналу на окремому виводі порту можна задати з допомогою команд встановлення лог. 0 (CBI PORTx.Y) і лог.1 (SBI PORTx.Y). Вхідний сигнал на окремому виводі порту можна перевірити, використовуючи команди умовного переходу SBIC PINx.Y або SBIS PINx.Y, які передбачають пропуск наступної команди по нульовому або одиничному значенні Px.Y.

2. Практична частина

2.1. Створення проектів в середовищі Proteus

Інтегроване середовище Proteus/Isis дозволяє моделювати роботу запрограмованого МК із зовнішніми пристроями. У даному випадку такими пристроями будуть світлодіоди і вимикачі. Програма запускається вибором Пуск/Всі програми/Proreus Professional/Isis Professional. В результаті появляється вікно редактора схем з координатною сіткою на якому можна розміщувати цільовий МК і зовнішні схемотехнічні елементи. Новий проект створюється вибором File/New design/Default. Для вибору цільового МК та інших елементів натискається кнопка вибору з бібліотек (зверху, зліва) P (pick from libraries). МК ATmega8515 вибирається із списку, отриманого вибором Microprocesors IC, резистори – Resistors/Resistor network/RES16DIPIS, світлодіоди – Optoelectronic/Leds/Led-Red, вимикачі – Switches &

Relays/Switches/Switch, джерело живлення – Miscellaneus/Cell, вимірювальні прилади – Vitual instruments mode. Елемент заземлення вибирається із списку отриманого вибором Terminal mode/Ground (восьма зверху кнопка на вертикальній лінійці інструментів зліва). Виводи елементів схеми з'єднуються між собою в режимі Selection mode (перша зверху кнопка на вертикальній лінійці інструментів). Для цього стрілка вказівник наводиться на відповідний вивід елементу і переміщується до іншого потрібного виводу. При переміщенні стрілки вказівника автоматично прокладається провід з'єднання. Для закріплення провідника у заданих точках потрібно клацати правою кнопкою миші.

Розроблений проект зберігається вибором File/Save Design As. В проект добавляється налагоджена в AVR Studio асемблерна програма (з розширенням *.asm), яка керуватиме роботою МК (вибір Source). Цільовий МК і нова програма вибираються у вікні, яке показане на рис. 12. Перед включення в проект нової програми потрібно вилучити з проекту стару програму Source/Add remove source files/Remove. Нова програма добавляється в проект вибором Source/Add remove source files/New. Після добавлення і збереження програми в проекті, схема запускається на виконання вибором Debug/Start або Debug/Execute.

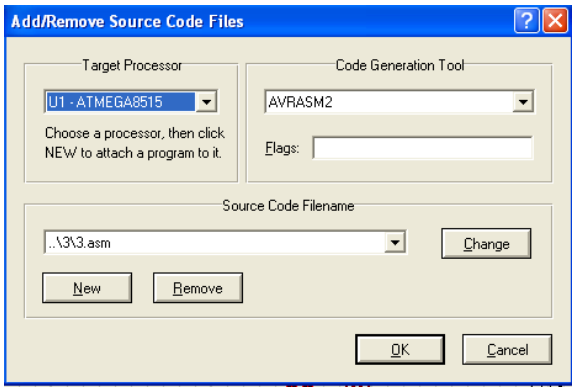


Рисунок 12 – Вікно добавлення asm програми в проект середовища Proteus

В середовищі Atmel Studio для завантаження hex-файлу в проект потрібно клацнути правою кнопкою миші на графічному зображенні МК і вказати шлях до hex-файлу. Після цього запуск на виконання здійснюється вибором з меню Debug/Start VSM Debugging і Debug/Run Simulation.

2.2. Приклади проектів у середовищі Proteus

Приклад1. Програма керування одним світлодіодом на 5-виводі порту B. Електрична схема з'єднань, виконана в середовищі Proteus/Isis показана на рис. 13.

```
; ****************************************************
; 1.asm - програма включення/виключення 1-го світлодіода
;****************************************************
;.include "m168pdef.inc"
.org 0x0000
   rjmp start
start:
    ldi r16, 0          ; перевстановлення статусу системи
    out SREG, r16       ; ініціалізація вказівника стеку
    ldi r16, low(RAMEND)
    out SPL, r16
```

```
    ldi r16, high(RAMEND)
    out SPH, r16

    sbi DDRB, 5       ;5, OUTPUT;
;ldi r17,0xFF
;out DDRB,r17

_loop:
    sbi PORTB, 5    ;включити LED
    rcall _delay
    cbi PORTB, 5    ;виключити LED
    rcall _delay
    rjmp _loop
_delay:
    ldi r24, 0x00        ;ітерації 1 сек затримки
    ldi r23, 0xd4
    ldi r22, 0x30
_d1:                     ;затримка ~1 сек
    subi r24, 1
    sbci r23, 0
    ;sbci r22, 0
    brcc _d1
    ret
```

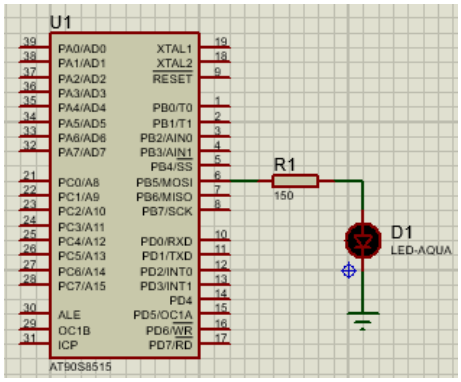


Рисунок 13 – Схема під’єднання світлодіоду для прикладу 1

Приклад 2. В програмі лінії порту В використовуються для статичної індикації, тому проініціалізовані як 0xFF (1111111) на вивід. Електрична схема з’єднань, виконана в середовищі Proteus/Isis показана на рис. 14.

```
; ********************************************************
; 2.asm - програма включення світлодіодів з парними номерами
;********************************************************
;.include "m8515def.inc"
;.include "m8def.inc"
.def Temp = r16
; параметри STK500: Режим=ISP, Fisp=57.6 kHz, HW=460.8 KHz
;***** Ініціалізація ***************************
.org 0x0000
    ldi Temp,0xFF    ; запис 1111111 в Temp
    out DDRB,Temp    ; встановити PORTB на вихід
;**** тест введення/виведення
    sbi PORTB,0      ;записати в біт 0 значення 1
                     ; через 1 такт встановлюється біт в PINB
    sbi PORTB,2      ;записати в біт 2 значення 1
    sbi PORTB,4      ;записати в біт 4 значення 1
    sbi PORTB,6      ;записати в біт 6 значення 1
```

```
loop: rcall loop    ;нескінчений цикл
```

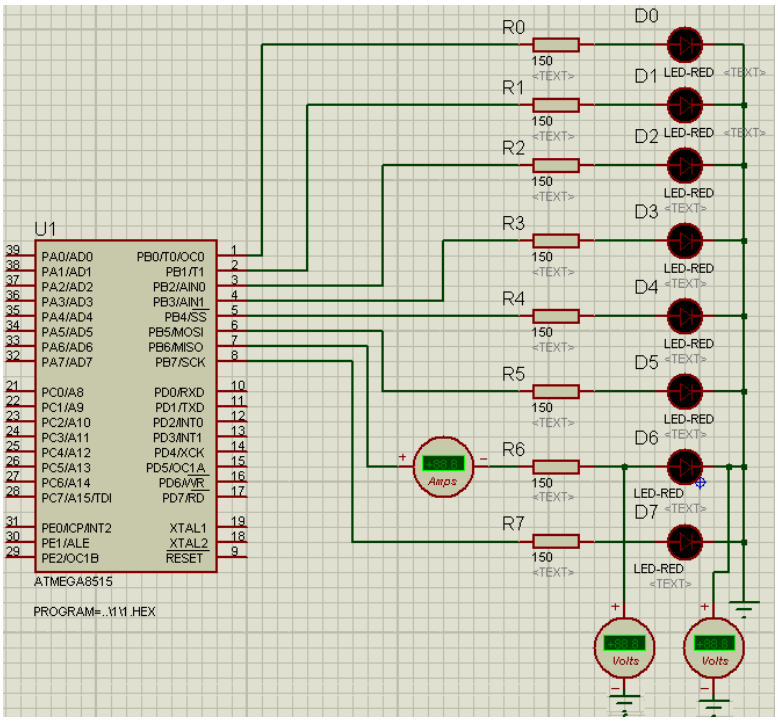


Рисунок 14 – Схема з’єднання світлодіодів для прикладу 2

Приклад 3. В програмі лінії порту В використовуються для індикації перемикання світлодіодів. Електрична схема з’єднань, виконана в середовищі Proteus/Isis показана на рис. 15.

```
     ; ********************************************************
     ; 3.asm – перемикання світлодіодів з декрементом лічильника
     ; ********************************************************
     .include "m8515def.inc" ;визначення фізичного МК
     .def Temp = R16 ; призначення символічного імені регістру r16
     .org 0x0000     ; розмістити код в пам'яті програми з адреси 0x0000
Loop1:
     ldi Temp,0xFF   ; завантажити значення 0xff в регістр Temp (ldi r16 по r31 !)
     out DDRB, Temp  ; задати напрямок передачі порту B - на вивід
Loop2:
     out PORTB, Temp ; вивести в порт B дані з регістра Temp
     dec Temp        ;  зменшити на 1 вміст регістра Temp
     cpi Temp,0
     breq Loop1
     ldi r17,0xff    ; затримка підбирається для кожної моделі ПК
d1:  ldi r18,0xff
d2:  dec r18
     brne d2         ; перехід по не дорівнює
     dec r17
     brne d1
     rjmp Loop2
```

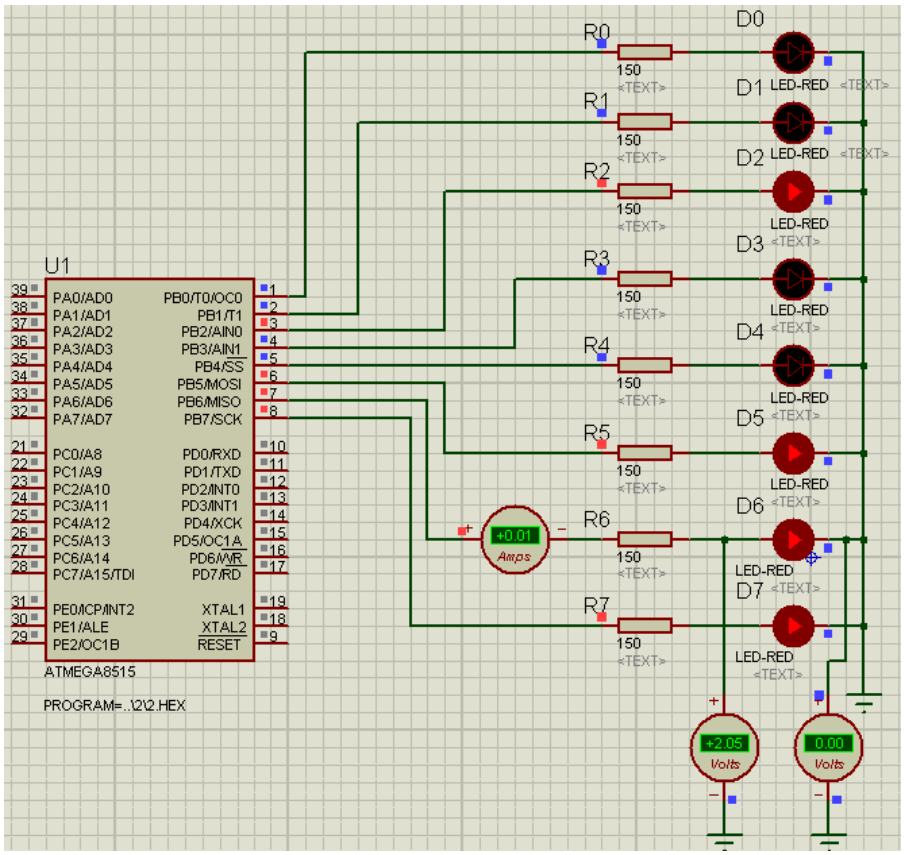


Рисунок 15 – Схема з’єднання світлодіодів для прикладу 3

Для зупинки і запуску перемикання світлодіодів можна використати розміщені на платі STK500 набори кнопок SW0-SW7. Кнопки під’єднані до джерела живлення VTG через резистор R=10 Ком. У розімкненому стані на виводі кнопки Swn встановлюється логічна 1. При натисканні кнопки на виводі SWn встановлюється логічний 0. З’єднавши вивід кнопки SWn з одним із виводів порту PORTx можна аналізувати її стан. Резистор 150 ом обмежує струм через МК. В МК AVR можна використовувати внутрішні підтягуючі резистори замість зовнішнього резистора 10 Ком. Для цього потрібно встановити напрям передачі виводу на вхід DDRX.n=0 і PORTX.n=1.

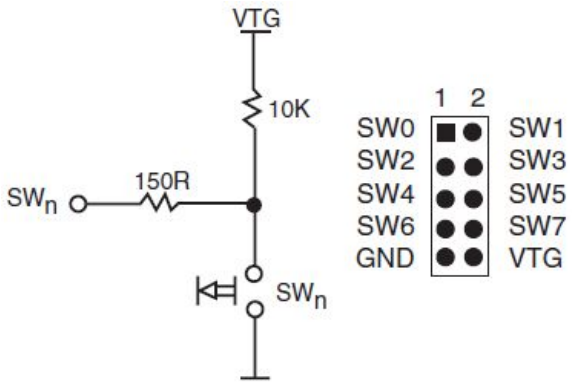


Рисунок 16 – Схема підключення кнопки в STK500

Схема з’єднання МК із зовнішніми елементами, яка залежить від логіки програми, для прикладу 4, показана на рис. 16

Приклад 4.

```
;***********************************************************************
; 4.asm - послідовне перемикання світлодіодів із зупинкою
; SW0 (START) запуск перемикання світлодіодів
; SW1 (STOP)  зупинка перемикання
; параметри STK500: Режим=ISP, Fisp=115.2 kHz, HW=526.6 KHz
;***********************************************************************
.include "m8515def.inc"    ;файл визначень для ATmega8515
.def temp = r16       ;тимчасовий регістр
.def reg_led = r20          ;регістр стану 8-ми світлодіодів
.equ START = 0        ;0-й розряд порту PD
.equ STOP = 1               ;1-й розряд порту PD

.org $000
          rjmp INIT
;***Ініціалізація***
INIT: ldi reg_led,0xFE      ;завантаження 1111_1110 в reg_led.0 для перемикання LED0
          sec         ; встановлення прапора C=1 в регістрі SREG
          set         ; встановлення прапора напрямку T=1 в регістрі SREG
          ser temp    ;ініціалізація temp значенням FF
          out DDRB,temp    ; порт PB на виведення 1111_1111
          out PORTB,temp   ;погасити світлодіоди ?
          clr temp         ;чистка регістра temp
          out DDRD,temp    ; порту PD на введення 0000_0000
          ldi temp,0x03    ;включення підтягуючих
          out PORTD,temp   ; резисторів до виводів порту PD, 0000 0011
WAITSTART:                  ;очікування
          sbic PIND,START  ; пропустити наступну команду, якщо PIND.0=0
          rjmp WAITSTART   ; лог.1 не натиснута кнопка START
LOOP:  out PORTB,reg_led    ; лог.0 натиснута кнопка START, включити СД
                           ; Затримка (два вкладених цикли)
          ldi r17,0xff      ;  підбирається для кожної моделі ПК !
d1:       ldi r18,0xff
d2:       dec r18
          brne d2           ; перехід, якщо прапор Z не дорівнює 0
          dec r17
          brne d1
          sbic PIND,STOP   ;пропустити наступну команду, якщо PIND.0=1
          rjmp MM          ;лог.1 - не натиснута кнопка START - перехід на MM
          rjmp WAITSTART   ;лог.0 - натиснута кнопка START, перехід на WAITSTART
MM:   ser temp             ;ініціалізація значенням FF
          out PORTB,temp   ;на вивід світлодіодів
          brts LEFT        ;перехід, якщо прапор T встановлений
          sbrs reg_led,0   ;пропуск наступної команди,
                           ;якщо 0-й розряд в регістрі reg_led встановлений
          set              ;T=1 - переключення прапора напрямку
          ror reg_led      ;зсув reg_led вправо на 1 розряд через біт Carry
          rjmp LOOP
LEFT:   sbrs reg_led,7  ; пропуск наступної команди, якщо
                        ; 7-й розряд регістра reg_led встановлений
        clt             ;T=0 - переключення прапора
                        ; напрямку
        rol reg_led     ; зсув reg_led вліво
        rjmp LOOP
```

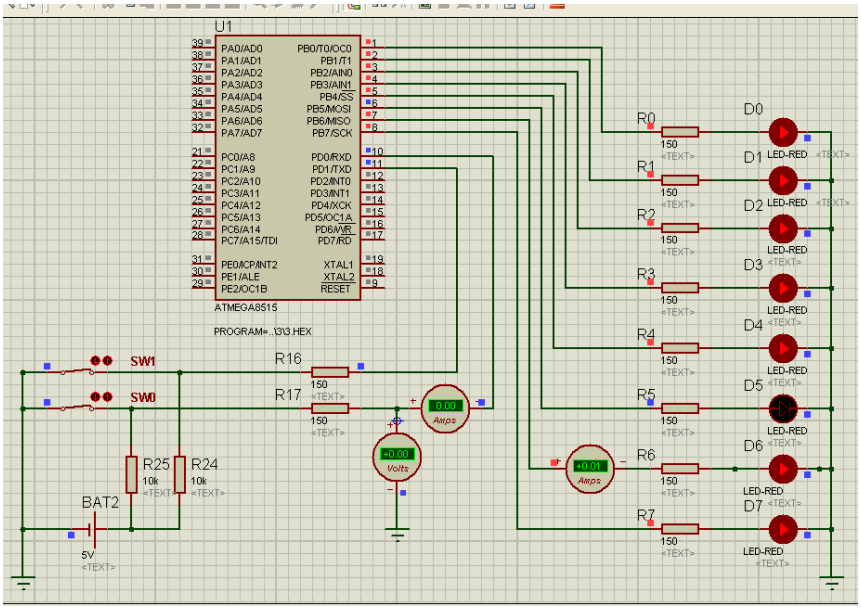


Рисунок 16 – Схема з’єднання світлодіодів і кнопок для прикладу 4

2.3. Керування світлодіодом кнопкою

Кнопки бувають декількох видів: тактові і фіксуючі, рис. 17. У тактовій кнопці при натисканні – контакти замикаються, при відпусканні – розмикаються; у фіксуючій кнопці – фіксується стан: при натисканні – контакти замикаються, при повторному натисканні – розмикаються.

а б

Рисунок 17 – Кнопки: а) тактова; б) фіксуюча

Схема підключення кнопки до МК показана на рис. 18.

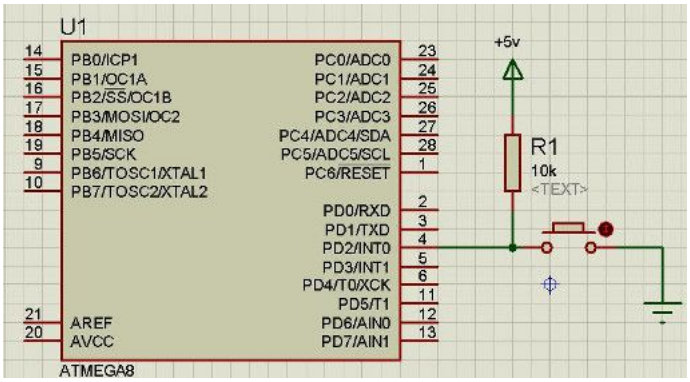


Рисунок 18 – Підключення кнопки до МК

На вивід PD.2 подається логічна одиниця (+5 В). При натисканні кнопки вивід PD.2 заземлюється і переходить в стан логічного нуля. Ці зміни можна детектувати і керувати станом виводу PD.7.

МК мають внутрішні підтягуючі резистори, тому зовнішній резистор R1 можна не використовувати. Щоб включити внутрішній підтягуючий резистор потрібно порт сконфігурувати на вхід `DDRD=0b00000000` і при ініціалізації порту в регістрі PORTD встановити в одиницю біт на якому знаходиться кнопка, `PORTD=0b00000100`. Цей вивід стає джерелом струму.

Якщо вивід налаштувати як вихід то:

- якщо на виводі логічний нуль то нічого не трапиться;
- якщо на виводі логічна одиниця то при натисканні кнопки просто закоротиться вивід на землю і через нього потече великий струм, який зруйнує вивід (струм через вивід не повинен перевищувати 40 мА, а струм від живлення Vcc до землі Ground не повинен перевищувати 200 мА).

Тому для захисту мікроконтролера між виводом і кнопкою потрібно поставити резистор 330 Ом, рис. 19. Для захисту світлодіоду потрібно поставити резистор R2 між виводом PD.7 і світлодіодом. Величина резистора розраховується в залежності від допустимого струму через світлодіод.

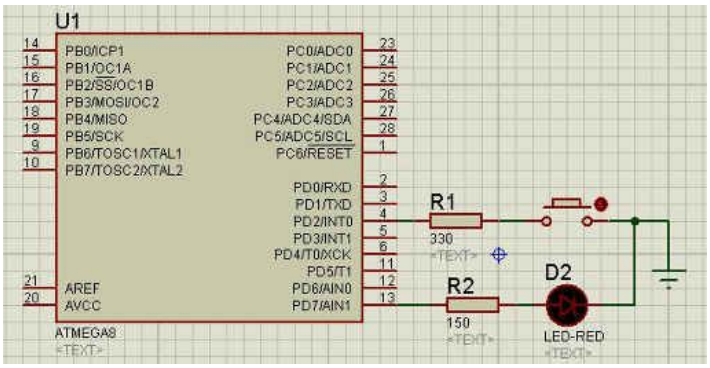


Рисунок 19 – Підключення кнопки і світлодіоду до МК

// Приклад 5. Програма перемикання роботи світлодіоду на мові Сі.

```
#include <avr/io.h>
#define F_CPU 100000UL  // 0.1 МГц
#include <util/delay.h>
void main(void)
{
// ініціалізація порту D
PORTD=0b00000000;
DDRD= 0b10000000;
while (1)
{
      if (PIND & 0b00000100) /* перевірка логічного рівня на PD.3, знак & - побітове І
      якшо PIND=0b00000100, тоді 0b00000100 & 0b00000100 => 0b00000100, тобто true
      якшо PIND=0b0000000, тоді 0b00000000 & 0b00000100 => 0b00000000, тобто false */
      PORTD=0b10000000; // PD.7=0
      else
      PORTD=0b00000100; // PD.7=1
};
delay_ms(100); // затримка 100 мсек захист від брязкоту контактів
}
```

// Приклад 6. Програма перемикання роботи світлодіоду на асемблері.

```
; перемикання LED натисканням кнопки + затримка від дренькоту
start:
ldi r16,0
out SREG,r16    ; reset system status

ldi r16, low(RAMEND)   ; встановлення стеку
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16

ldi r17,(1<<7)   ; 0b0001_0000
out DDRD, r17    ; DDRD (PD.7 -> out) LED
ldi r16,(1<<2)   ; 0b0000_0100 (PD.2 <- in) кнопка

main:           ; основний цикл
out PORTD,r16   ; DDRD.2=0, PD2=1 включення підтягуючого резистора
sbic PIND, 2    ; якщо PD2=0 (кнопка натиснута), пропустити наступний рядок
rjmp main
rcall delay     ; call затримка на дренькіт контактів

wait:           ; цикл очікування, поки натиснута кнопка
sbis PIND, 2    ; якщо PD2=1 (кнопка відпущена), пропустити наступний рядок
rjmp wait
rcall delay     ; виклик підпрограми затримки на дренькіт контактів
sbic PIND, 2    ; якщо PD2=0 (кнопка натиснута), пропустити наступний рядок
eor r16,r17     ; виключальне АБО r16 і r17, результат в r16
                ; r16 = 0b0000_0100
           ; r17 = 0b0001_0000
           ; eor-> r16 = 0b0001_0100    перемикає LED
rjmp main

delay:            ; підпрограма затримки
ldi r18, 255      ; встановлення величина затримки в циклах r18 * r19
ldi r19, 25
del:              ; реалізація затримки
subi r18, 1       ; відняти 1 від r18
sbci r19, 0       ; відняти 0 і carry від r19
brcc del          ; якщо не було перенесення повернутися до позначки del
ret
```

Завдання.

1. В AVR Studio/Microchip Studio/Atmel Studio 7 відкомпілювати і проаналізувати в покроковому режимі програму прикладу 2. Виконати програму (2.hex) на платі STK500 з цільовим МК.

Примітка. При наявності плати STK500 виконати наступні пункти. Отриманий після успішної трансляції програми hex-файл використати для програмування цільового МК. При програмуванні слідкувати, щоб тип цільового мікроконтролера, встановленого на використовуваній платі, співпадав з типом мікроконтролера в полі Program. В процесі програмування у вікні STK500 появляються повідомлення про хід завантаження програми.

Переконавшись у правильності завантаження по виведених повідомленнях, перевірити роботу програми на макеті. Для цього вимкнути живлення STK500, 10-провідним шлейфом з’єднати виводи роз'єму відповідного порту з виводами роз’єму світлодіодів. Подати живлення і перевірити роботу завантаженої програми.

2. В AVR Studio/Microchip Studio/Atmel Studio 7 відкомпілювати і проаналізувати в покроковому режимі програму прикладу 3. Виконати програму (3.hex) на платі STK500 з цільовим МК.

3. В AVR Studio/Microchip Studio/Atmel Studio 7 відкомпілювати і проаналізувати в покроковому режимі програму прикладу 4. Виконати програму (4.hex) на платі STK500 з цільовим МК.

Примітка. Симуляція замикання і розмикання кнопок START і STOP здійснити встановленням 0 (білий колір) і 1 (чорний колір) в маленьких квадратиках порту ліній інтерфейсу PIND. Перед початком роботи програми встановити для обох кнопок стан логічної 1 (кнопки відтиснуті).

Переконавшись в правильності роботи програми, змінити параметри циклів затримки, щоб тривалість затримки склала 0,5 с. Перевірити час затримки. Для цього встановити контрольні точки (Debug/Toggle Breakpoint) перед початком виконання програмного блоку затримки і після виходу з нього. Запустити програму в режимі прогону (Debug/Run) із зупинкою в контрольних точках, оцінити час затримки, контролюючи або значення лічильника циклів Cycle Counter у вікні Workspace AVR Studio 4 (закладка I/O, секція Processor), або значення Stop Watch.

4.1. Підібрати значення величин, які завантажуються в регістри r17, r18 такими, щоб затримка становила 0,5 с.

4.2. Модифікувати програму прикладу 4 для включення і виключення світлодіодів по одній із заданих послідовностей:

№	Послідовності	Кнопка SW0	Кнопка SW1
1	**7-6-5-4-3-2-1-0**-1-…	зупинити	запустити
2	**7-6-5-4-3-2-1-0**-7-…	запустити	зупинити
3	**7-5-3-1-**7-…	зупинити	запустити
4	**7-5-3-1-**1-…	запустити	зупинити
5	**6-4-2-0**-6-...	зупинити	запустити
6	**6-4-2-0**-2…	запустити	зупинити
7	**7-7-6-6-5-5-4-7**…	зупинити	запустити
8	**7-7-6-6-5-5-4-5**…	запустити	зупинити
9	**7-7-5-5-3-3-1-7**…	зупинити	запустити
10	**7-7-5-5-3-3-1-1**…	запустити	зупинити
11	**7-0-6-1-5-2-4-3**-7-…	зупинити	запустити
12	**7-0-6-1-5-2-4-3**-4-…	запустити	зупинити
13	**76-54-32-10-**76-..	зупинити	запустити
14	**76-54-32-10-**32-…	запустити	зупинити
15	**765-654-543-432-321-**…	зупинити	запустити
16	**765-654-543-432-765-**…	запустити	зупинити
17	**7654-3210**-7654-…	зупинити	запустити
19	**7654-3210**-3210…	запустити	зупинити
20	**7531-6420-**7531…	зупинити	запустити

5. Написати програму на асемблері для керування одною кнопкою двома світлодіодами (один вмикається, а інший вимикається).

Примітка. Номер варіанта завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Блок схема алгоритму програми.
4. Текст програми із поясненнями.
5. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми

Питання.

1. Для чого призначена плата STK500 і які її основні елементи.
2. Намалювати і пояснити схеми підключення кнопок і світлодіодів до VR.
3. Призначення і особливості інтерфейсу STK500 в AVR Studio/Microchip Studio/Atmel Studio 7.
4. Паралельні порти введення/виведення і їх регістри.
5. Яку функцію виконують вбудовані підтягуючі резистори? Як можна підключити і відключити ці резистори?
6. Як задається альтернативна функція розряду порту?
7. В яких станах може знаходитися регістр даних PORTx?
8. В яких станах може знаходитися регістр напрямку DDRx?
7. Яким сигналом (логічний 0 або логічна 1) можна включити світлодіод?
8. Як визначити потенціал виводу мікросхеми PINx.Y?
9. Як визначити величину програмної затримки і як збільшити її точність?
10. Для чого призначений симулятор Proteus.
11. Як в середовищі Proteus запустити на виконання асемблерну програму?

ЛАБОРАТОРНА РОБОТА № 6. Внутрішні і зовнішні переривання.

Мета роботи – вивчення внутрішніх і зовнішніх переривань та їх оброблення, підключення кнопок, клавіатур, світлодіодів.

1. Переривання і їх оброблення

Основним завданням МК є очікування і оброблення деяких подій. Найбільш універсальний спосіб оброблення подій – організація основної програми як нескінченого циклу. Всередині цього циклу деяким способом відслідковується поява подій. При виникненні події встановлюється значення одного біту (прапора) у відведеному регістрі, комірці пам'яті або навіть безпосередньо стан виводу МК. Основна програма в циклі перевіряє значення потрібних прапорів, і при зміні одного з них переходить до відповідної процедури оброблення події.

Як альтернатива універсальному способу відслідкування подій в сучасних МК використовуються апаратні переривання (interrups). При виникненні деякої події тут також встановлюються деякі біти прапори, але перехід до їх оброблення здійснюється апаратно. Програмісту потрібно лише написати підпрограму оброблення переривань. У деяких випадках основна програма може містити один рядок нескінченого циклу – `loop: rjmp loop`, все інше буде виконуватися через переривання.

Прапори переривань утворюють ієрархію. Зверху ієрархії знаходиться біт I регістра SREG, який дозволяє (лог.1, команда `sei`) або забороняє (лог.0, команда `cli`) любі апаратні переривання. За замовчуванням прапор I очищений і при запуску МК переривання заборонені. Для кожного конкретного переривання є свій дозволяючий/забороняючий біт у відповідному регістрі (наприклад, для таймерів це регістри TIMSK, для зовнішніх пристроїв – GIMSK). При використанні переривань ці біти потрібно встановлювати в стан лог.1.

Загальна схема оброблення переривань наступна. При виникненні любого переривання апаратно очищається прапор I регістра SREG. Він автоматично встановлюється знову, коли закінчиться оброблення переривання (за командою `reti`). Цей прапор можна встановлювати програмно в програмі-обробнику (безпосередньо або командою `sei`), дозволивши вкладені переривання. Після скидання прапора I МК визначає, запит на оброблення якого саме переривання відбувся, – це встановлюється за прапором конкретного переривання (для таймерів – в регістрі TIFR, для зовнішніх переривань – в регістрі GIFR). Після визначення типу переривання МК автоматично обчислює адресу відповідного вектора переривань, які звичайно розміщуються за початковими адресами пам'яті програм. За цією адресою має знаходитися команда `rjmp` (або `jmp` для МК з розміром пам'яті 16 Кбайт і більше), яка містить адресу підпрограми обробника.

Перед тим як перейти за вектором переривання, МК скидає прапори переривання і автоматично зберігає вміст лічильника команд у стеку. Тому для успішного виклику переривань в програму ініціалізації МК необхідно добавити наступні рядки (їх звичайно ставлять після позначки, на яку здійснюється перехід за вектором скидання RESET)

```
RESET:
ldi temp,low(RAMEND)   ; завантаження вказівника стеку
out SPL, temp
ldi temp,high(RAMEND)   ; завантаження вказівника стеку
out SPH, temp
. . .    ; інші команди
```

```
sei      ; дозвіл переривання
```

Підпрограма обробник повинна закінчуватися командою виходу з переривання `reti`. За цією командою відновлюється лічильник команд і знову встановлюється загальний прапор дозволу переривань.

2. Внутрішні і зовнішні переривання

В МК всі переривання апаратні і діляться на внутрішні і зовнішні. Внутрішні переривання можуть виникнути від любого внутрішнього функціонального блоку: таймерів, аналогового компаратора, послідовного порту. Внутрішнє переривання – це подія, яка виникає в системі і перериває виконання основної програми.

МК ATmega8515 може обробляти 3 зовнішні і 17 внутрішніх переривань. Як входи зовнішніх переривань використовуються порти з альтернативною функцією: PD2, PD3 – для переривань INT0, INT1 і PE0 – для переривань INT2. Запити зовнішніх переривань INT0, INT1 можуть бути подані низьким рівнем сигналу (L), переходом від високого рівня сигналу до низького (HL – по спадаючому фронту), переходом від низького рівня сигналу до високого (LH – по наростаючому фронту). Зовнішнє переривання 1 активується зовнішнім виводом INT1, якщо встановлені I-біт в SREG і відповідна маска в регістрі GICR (7-біт INT1, 6-біт INT0, 5-біт INT2). В регістрі управління МК MCUCR необхідно встановити біти ISC11, і ISC10 як в табл. 1.

Таблиця 1 – Рівні сигналів INT1 які генерують запит на переривання

ISC11	ISC10	Зміни сигналу
0	0	L
0	1	Любі логічні зміни HL/ LH
1	0	HL
1	1	LH

Зовнішнє переривання 0 активується зовнішнім виводом INT0, якщо встановлені I біт в SREG і відповідна маска в GICR. В регістрі керування МК MCUCR необхідно встановити біти ISC01, і ISC00 відповідно до табл. 2.

Таблиця 2 – Рівні сигналів INT0 які генерують запит на переривання

ISC01	ISC00	Зміни сигналу
0	0	L
0	1	Любі логічні зміни HL/LH
1	0	HL
1	1	LH

Асинхронне зовнішнє переривання 2 активується якщо зовнішнім виводом INT2, якщо встановлені I біт в SREG і відповідна маска в GICR. Для переривання INT2 необхідно встановити біт ISC2 в регістрі EMCUCR (extended MCUCR, 0-біт ISC2). При ISC2 = 0 спадаючий фронт (HL) на INT2 активує переривання, при ISC2 = 1 – наростаючий фронт (LH) активує переривання, табл. 3.

Таблиця 3 – Рівні сигналів INT2 які генерують запит на переривання

ISC2	Зміни сигналу
0	HL
1	LH

Переривання за переходами LH, HL, LH/HL на INT0/INT1 детектуються синхронно за тактовим сигналом.

Переривання за низьким рівнем L на INT0/INT1 і переривання за фронтами (HL, LH) на INT2 детектуються асинхронно. Тому асинхронне зовнішнє переривання використовується для виведення МК з режиму енергозбереження (sleep).

Усі моделі МК AVR мають два базових режими енергозбереження: Idle mode і Power Down mode. В режимі Idle mode (режим очікування) зупиняється основний процесорний модуль, а всі периферійні пристрої (таймери, АЦП, порти) продовжують функціонувати. У цьому режимі споживання енергії зменшується тільки на 30-50%.

У режимі Power Down mode (режим “глибокого” енергозбереження) зупиняються всі вузли МК, за винятком сторожового таймера (якщо він включений), системи обробки асинхронних зовнішніх переривань і модуля TWI. Вихід з цього режиму можливий за скиданням МК або сторожового таймера, або переривання від TWI. Споживання в цьому режимі може становити до декількох десятків мікроампер.

Приклад 1. Програма оброблення переривань при натисканні кнопок.

```
    ;.include "m8515def.inc"
    .CSEG
    .org  $000
    rjmp   RESET     ;Reset Handler
    .org  $001
    rjmp   EXT_INT0  ;IRQ0 Handler
    .org  $002
    rjmp   EXT_INT1  ;IRQ1 Handler

    .org  $010
    reti            ;Store Program Memory Ready Handler

;-------- Підпрограма переривання --------
EXT_INT0:
    sbic  PORTA, 0  ; пропустити наступну інструкцію, якщо PA.0 == 0
    rjmp  elsePA0
    sbi  PORTA, 0   ; встановити PA.0 = 1
    reti
elsePA0:
    cbi  PORTA, 0  ; встановити PA.0 = 0
    reti

EXT_INT1:
    sbic  PORTA, 1 ; пропустити наступну інструкцію, якщо PA.1 == 0
    rjmp  elsePA1
    sbi  PORTA, 1  ; встановити PA.1 = 1
    reti
elsePA1:
    cbi  PORTA, 1 ; встановити PA.1 = 0
    reti
;-----------------------------------------------------

    ; ініціалізація стеку
RESET:
```

```
    ldi  r16, Low(RAMEND)
    out  SPL, r16
    ldi  r16, High(RAMEND)
    out  SPH, r16

    ldi  r16, 0x00
    ldi  r17, 0xFF
    ; ініціалізація портів
    out  DDRA, r17      ; порт A на вивід
    out  PORTA, r16     ; на виводах нулі

    out  DDRD, r16      ;порт D на вхід для (INT1, INT0)
    ldi r16, 0b00001100
    out  PORTD, r16

  ; дозволити зовнішні переривання INT0, INT1
    ldi  r16, (1<<INT0)|(1<<INT1)  ;вибір переривання, побітове OR виразів (1<<6)|(1<<7) =>
1100_0000
    out  GICR,r16                  ; : 7 - INT1: 6 - INT0 : 5 - INT2 : ... : 0 :

    ; як сигнали активують переривання
    ldi  r16, (1<<ISC01) | (0<<ISC00)  |  (1<<ISC11) | (1<<ISC10) ;  : ... : 3 - ISC11 : 2 -
ISC10 : 1 - ISC01 : 0 - ISC00 :
    out  MCUCR, r16   ; INT1-по зрост. фронту, INT0-по спад. фронту   0000_ 1110

    ;скидування прапорів зовнішнього переривання
    ldi  r16, (0<<INTF0) | (0<<INTF1)
    out  GIFR, r16

    ;загальний дозвіл на переривання (I-біт в SREG)
    sei

main:
         rjmp  main
```

Електрична схема з’єднання МК із кнопками в середовищі Proteus/Isis, показана на рис. 1.

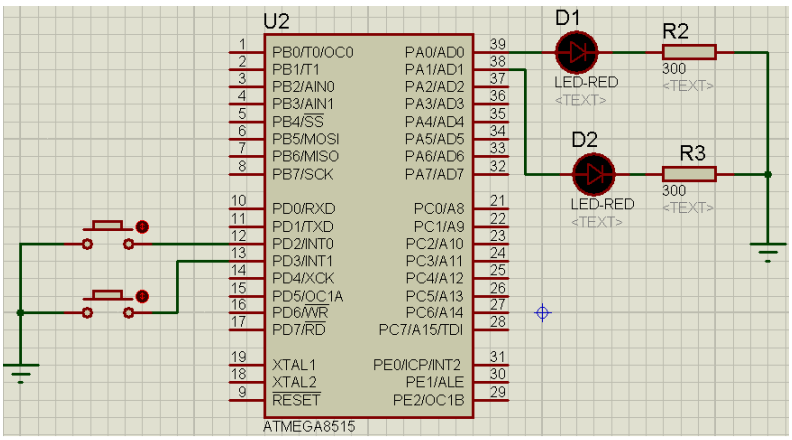


Рисунок 1 – Електрична схема з’єднання МК із кнопками

3. Обробка подій у нескінченому циклі

Обробка подій у нескінченому циклі може використовуватися для підрахунку натискання кнопки і висвітлення загальної кількості натискань у двійковому коді на світлодіодах. У

програмі враховується брязкіт кнопок. При натисканні або відпусканні кнопки генерується множина імпульсів, з яких потрібно вибрати один. Звичайно реагують не на натискання кнопки, а на відпускання. Послідовність виявлення відпускання кнопки наступна: спочатку виявляється натискання кнопки (низький рівень на виводі PINx), робиться перша пауза для пропуску брязкоту (0,2 сек), і потім виявляється відпускання (перший високий рівень на виводі PINx). Після виявлення відпускання, наприклад збільшується лічильник натискань і робиться друга пауза проти брязкоту (0,5 сек) і цикл повторюється спочатку. Величини 0,2 і 0,5 уточнюються в залежності від реальної конструкції кнопок.

Для відліку тривалості пауз необхідний імітатор таймера. Для формування часового інтервалу без таймера можна використати тривалість машинного такту. В МК AVR більшість команд виконується за один машинний такт. Тому для формування інтервалу в 1 сек при тактовій частоті 4 МГц потрібно виконати послідовність з чотирьох мільйонів однотактних команд. Для формування часових інтервалів використовуються програмні затримки з використанням команд декременту. Наприклад в три регістри записується одне число Roz2, Roz1, Roz0. Потім послідовно зменшується молодший розряд Roz0, коли його величина досягне нуля, зменшується на одиницю наступний Roz1 і здійснюється перехід на зменшення молодшого розряду із значення 255 (воно отримується автоматично при відніманні 1 від 0). Такі ітерації продовжуються до тих пір, поки всі розряди не стануть нулями. Послідовність команд, яка реалізує цей алгоритм показана в табл. 2, варіант 1. Недолік такої реалізації в тому, що кількість тактів є змінною, так як команда *brne* виконується за один такт, якщо умова не виконується і за два такти – якщо виконується. Реалізація з постійною кількістю тактів показана у варіанті 2, для якого число N, яке відповідає потрібному інтервалу часу T (сек), при тактовій частоті F (Гц), можна отримати за такою формулою $N=T \cdot F/5$. Так, при T=1 сек, F=4 МГц, N=1·4_000_000/5=800_000=0x0c3500.

Таблиця 2 – Значення затримок

К-ть 8-бітових розрядів	Значення, N	Затримка, T
1	$2^8 = 256$	0,32 мсек
2	$2^{16} = 65536$	81,92 мсек
3	$2^{24} = 16777216$	20,97 сек
4	$2^{32} = 4\ 294\ 967\ 296$	1,49 год

Таблиця 3 – Реалізації програмних затримок

Варіант 1	Тактів		Варіант 2	Тактів	
`Delay:`			`Delay:`		
`dec Roz0`	1		`subi Roz0,1`	1	Відняти від регістра 1
`brne Delay`	1,2	Перейти на Delay, якщо прапор Z != 1	`sbci Roz1,0`	1	Відняти від регістра 0 і прапор Carry
`dec Roz1`	1		`sbci Roz2,0`	1	

brne Delay	1,2		brcc Delay	2	Перейти на Delay, якщо прапор C == 0
dec Roz2	1				
brne Delay	1,2				
К-ть тактів	6÷9			5	

```
.def Roz0 = r17             ;розряд числа
.def Roz1 = r18
.def Roz2 = r19
;*** затримка ***
.cseg
.org $000

;  число для заданої затримки (t,сек) і частоти (f,Гц)   N=T*F/5
;  t=1 мсек,   f=1 МГц, число  N=0.001*1,000,000/5 =    200  = 0xc8
;  t=10 мсек,  f=1 МГц, число  N=0.01*1,000,000/5  =   2,000 = 0x07d0
;  t=100 мсек, f=1 МГц, число  N=0.1*1,000,000/5   =  20,000 = 0x4e20
;  t=1 сек,    f=1 МГц, число  N=1*1,000,000/5     = 200,000 = 0x030d40

                ldi Roz0,0x40
                ldi Roz1,0x0d
                ldi Roz2,0x03
                rcall Delay

reset:
                rjmp reset

;*** підпрограма затримки
Delay:
                subi Roz0,1     ; відняти 1
                sbci Roz1,0     ; відняти 0 і прапор Carry
                sbci Roz2,0     ; відняти 0 і прапор Carry
                brcc Delay      ; вихід з підпрограми, якщо прапор Z == 0
ret             ; вихід з підпрограми

; приклад ітерацій для числа 0x020202
;0x02 0x02 0x02
;0x02 0x02 0x01
;0x02 0x02 0x00
;0x02 0x01 0xff    c=1
;0x02 0x01 0xfd
;...
;0x02 0x01 0x00
;0x02 0x00 0xff   c=1
;...
;0x02 0x00 0x00
;0x01 0xff 0xff   c=1
;0x01 0xff 0xfd
;...
;0x01 0xff 0x00
;0x01 0xfd 0xff
;...
```

Приклад 2. Програма підрахунку натискань кнопки у двійковому коді.

```
;*****************************************************************
; 2.asm - підрахунок натискання кнопки у двійковому режимі
; частота зовнішнього кварцу 4 МГц
;*****************************************************************
;.include "m8515def.inc"    ;файл визначень для ATmega8515
; директиви
```

```
.def temp = r16            ;тимчасовий регістр
.def Roz0 = r17       ;розряди затримки
.def Roz1 = r18
.def Roz2 = r19
.def Counter = r20           ; лічильник
;*** Програма ***
.cseg
.org $000
rjmp reset
reset:
; ініціалізація внутрішньої периферії
      ldi temp, low(RAMEND)
      out SPL, temp
      ldi temp, high(RAMEND)
      out SPH, temp
                              ; при включенні порт налаштовується на вхід, DDRD=0x0
      ldi temp,0b00000100   ;2-й розряд порту D
      out PORTD,temp   ; підтягуючий резистор
        ldi temp,0b11111111
      out DDRB,temp         ; порт B на виведення
      clr Counter           ; чистка лічильника
; основний цикл програми
Pin_cykl:             ; відслідкування кнопки
      sbic PIND,2     ; пропустити наступну інструкцію, якщо pind.2 == 0, кнопка
натиснута
      rjmp Pin_cykl
        ; кнопка натиснута, пауза 0.2 с, N=0x027100
        ldi Roz2,0x02
        ldi Roz1,0x71
        ldi Roz0,0x00
        rcall Delay
Pin_off:            ; відслідкування відпускання кнопки
        sbis PIND,2 ; пропустити наступну інструкцію, якщо pind.2 == 1, кнопка розімкнута
        rjmp Pin_cykl
        inc Counter        ; якщо відпущена, збільшити лічильник
        out PORTB,Counter  ; виведення лічильника в порт
        ; кнопка розімкнута, пауза 0.5 с, N=0x061A80
        ldi Roz2,0x06
        ldi Roz1,0x1a
        ldi Roz0,0x80
        rcall Delay
        rjmp Pin_cykl

;*** підпрограма затримки ***
Delay:
        subi Roz0,1     ; відняти 1
        sbci Roz1,0     ; відняти 0 і прапор Carry
        sbci Roz2,0     ; відняти 0 і прапор Carry
        brcc Delay      ; перейти на Delay, якщо Z == 0
ret                     ; повернення з підпрограми
```

Електрична схема з’єднання МК із кнопкою і світлодіодами, якими керує програма в середовищі Proteus/Isis, показана на рис. 2.

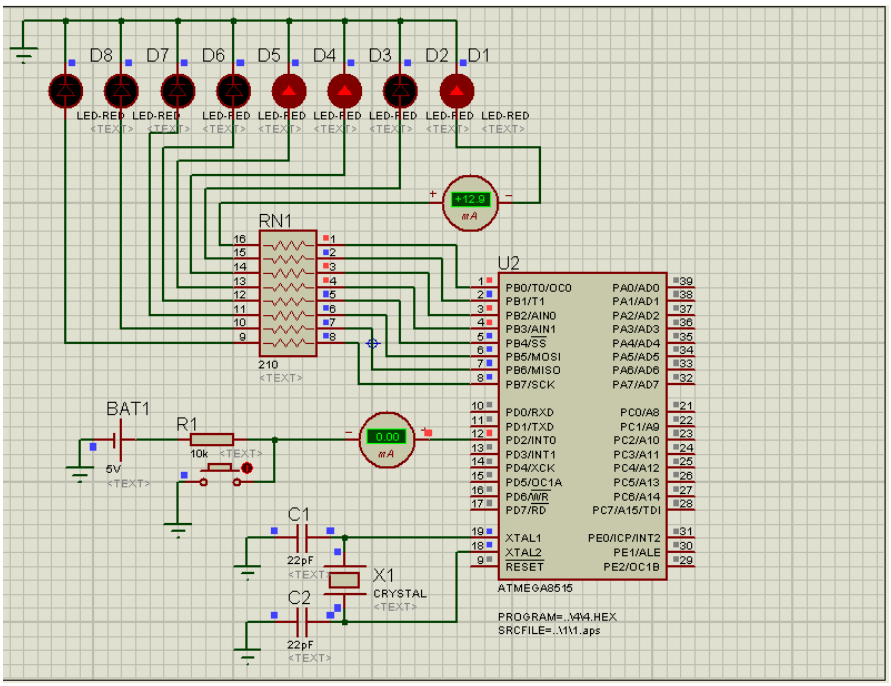


Рисунок 2 – Електрична схема з’єднання МК із кнопкою і світлодіодами

В прикладі 3 обробляється кнопковий регістр із індикацією стану кнопок SWx (x=0,1,2,3) короткотривалим включенням відповідного світлодіода, а також включення і виключення світлодіодов LEDx при повторному натисканні кнопок SWx (x = 4, 5, 6, 7).

В прикладі використовується макровизначення для перевірки стану кнопок. При кожному натисканні кнопки значення відповідного біта в слові стану молодшої і старшої груп кнопок (st_L, st_H) міняється на протилежний (командою `eor`) і зберігається. Для виведення на світлодіоди формується байт виводу reg_led. Програма виконується в середовищі Proteus/Isis, електрична схема з’єднання кнопок і МК показана на рис. 3.

Приклад 3.

```
;*****************************************************************
; 3.asm - оброблення кнопочного регістра
; Опитування і індикація натискання кнопок SWx, (x=0,1,2,3)
; короткочасним включенням LEDx.
; Включення  і виключення LEDx (x=4,5,6,7) після повторного
; натискання кнопок SWx, (x=4,5,6,7).
; параметри STK500: Режим=ISP, Fisp=57.6 kHz, HW=460.8 KHz
;*****************************************************************
;.include "m8515def.inc"    ;файл визначень для ATmega8515
; директиви
.def temp = r16        ;тимчасовий регістр
.def st_L = r17        ;стан молодшої групи SW_0123
.def st_H = r18        ;стан старшої  групи SW_4567
.def sw_cod = r22      ;код стану замкнутої кнопки
.def reg_led = r23     ;регістр стану світлодіодів

; макровизначення
.macro  testH_sw
      sbic PIND, @0    ;пропустити наступну інструкцію, якщо біт @0==0
      rjmp quit        ; і встановлення
      bld sw_cod, @0   ; біту замкнутої кнопки
      eor st_H,sw_cod  ;перемикання біта стану
      mov reg_led,st_H      ;байт для індикації
```

```
      com reg_led           ; доповнення до 1, інвертування для виводу

      out PORTB,reg_led     ; на СД
      clr reg_led           ; чистка
      clr sw_cod
      rcall DELAY_0         ;затримка
;пропустити наступну інструкцію, якщо біт @0==1,кнопка відпущена?
wait: sbis PIND, @0
      rjmp wait
quit: nop
.endmacro

.macro  testL_sw
;пропустити наступну інструкцію, якщо біт @0==0,
; перевірка кнопки SWx(x=0,1,2,3)
      sbic PIND,@0
      rjmp quit       ; і встановлення
; Завантажити біт T=1 у біт @0 рег. sw_cod, біта замкнутої кнопки
 bld sw_cod,@0
 eor st_L,sw_cod ;перемикання біта стану
      mov reg_led,sw_cod    ;байт
      or reg_led,st_H ; об'єднання для індикації
      com reg_led           ;доповнення до 1, інвертування для виводу на СД

      out PORTB,reg_led     ; на СД
      clr sw_cod
      rcall DELAY_0         ;коротка затримка
      ori reg_led,0x0f     ; перед гасінням
      out PORTB,reg_led    ; світлодіодів молодшої групи
quit:      nop
.endmacro

; *** програма ***
.cseg
.org $000
      rjmp init
;Ініціалізація внутрішньої периферії
init: ldi temp,low(RAMEND)  ;встановлення
      out SPL,temp          ; вказівника стеку
      ldi temp,high(RAMEND) ; на останню
      out SPH,temp          ; комірку ОЗП

      ser temp         ;налаштування 0xFF -> temp
      out DDRB,temp    ; порту PB на виведення
      out PORTB,temp   ; запис у порт B лог. 1
      clr temp         ; чистка temp
      out DDRD,temp    ;     ; налаштування порту PD на вхід
      ser temp         ; 0xFF -> temp
      out PORTD,temp   ; запис у порт D лог. 1
      clr sw_cod       ; чистка коду кнопки
      clr st_L         ; чистка операнда
      clr st_H
      clr reg_led
      set              ;T=1

; основний цикл програми
input:     testL_sw 0
      testL_sw 1
      testL_sw 2
      testL_sw 3
      testH_sw 4
      testH_sw 5
      testH_sw 6
      testH_sw 7
```

```
      rjmp input

; *** Підпрограми затримок ***
; Затримка коротка
DELAY_0:   ldi r21,255
d0:   dec r21
      brne d0
      ret

; Затримка довга
DELAY:     ldi r19,10   ; Величина затримки підбирається для кожного ПК!

d1:   ldi r20,255
d2:   ldi r21,255
d3:   dec r21
      brne d3
      dec r20
      brne d2
      dec r19
      brne d1
      ret
```

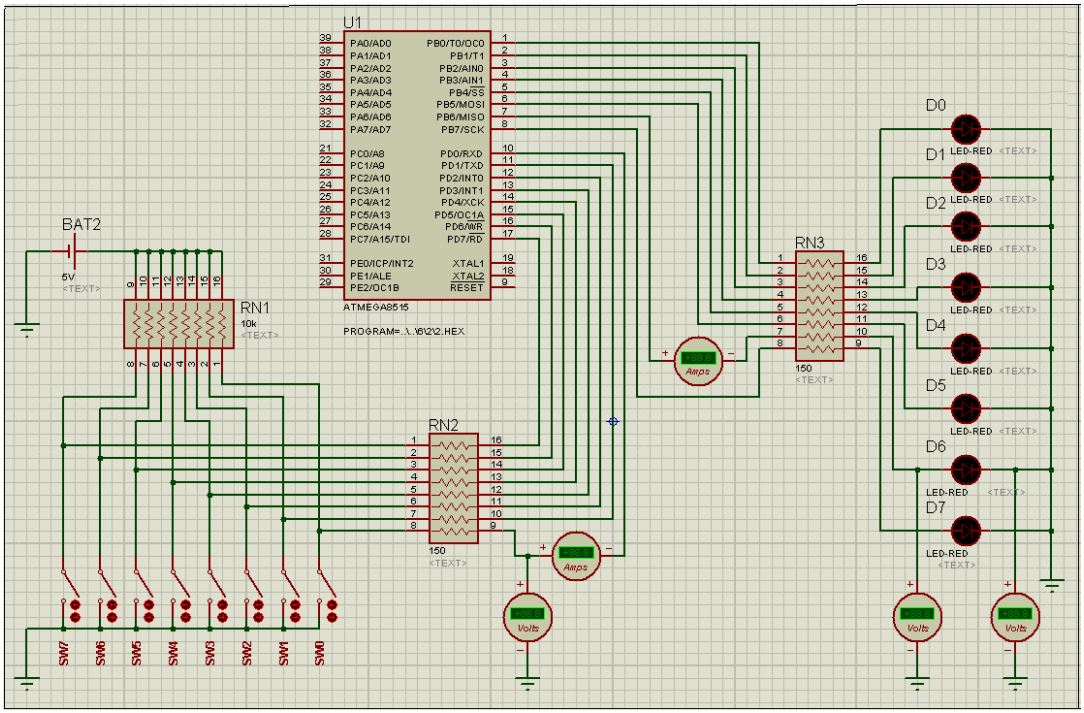


Рисунок 3 – Оброблення кнопкового регістра

4. Використання зовнішніх переривань для перемикання світлодіодів

В програму перемикання світлодіодів (лабораторна робота 5, приклад 3) при використанні зовнішнього переривання від кнопки STOP в блок ініціалізації МК вноситься ряд змін:

- додається вектор переривань;
- вказівник стеку встановлюється на останню комірку ОЗП;
- дозволяється зовнішнє переривання INT0 (за сигналом 0 на лінії 2 порту PD) і загальне взагалі.

Оскільки зовнішнє переривання INT0 подано сигналом на вході порту PD2, як кнопка STOP використовується SW2 і PD2 програмується на введення. Затримка забезпечується

підпрограмою DELAY. Переривання викликається кнопкою STOP. Програма виконується в Proteus/Isis, електрична схема з'єднання кнопок і МК показана на рис. 4.

Приклад 4.

```
;*********************************************************************
; 4.asm - перемикання світлодіодів (СД) з використанням переривань
; натискання кнопки SW2 (START) - запуск і відновлення перемикання СД
; натискання кнопки SW1 (STOP) - зупинка перемикання
;*********************************************************************
;.include "m8515def.inc"    ;файл визначень для ATmega8515
.def temp = r16             ;тимчасовий регістр
.def reg_led = r20          ;стан регістра світлодіодів
.equ START = 0              ;0-й розряд порту PD

.org $000
     ;***Вектори переривань***
     rjmp INIT              ;обробка скидання
     rjmp STOP_PRESSED          ;обробка зовнішнього переривання INT0(STOP)
;***ініціалізація МК***
INIT: ldi reg_led,0xFE
          ldi temp,low(RAMEND)    ;встановлення
          out SPL,temp          ; вказівника стека
          ldi temp,high(RAMEND) ; на останню
          out SPH,temp          ; комірку ОЗП

          sec              ;C=1  встановлення прапорів
          set              ;T=1
          ser temp         ;ініціалізація DDRB=0xff
          out DDRB,temp         ; порт PB на виведення
          out PORTB,temp   ;засвітити СД
          clr temp         ;ініціалізація DDRD = 0x00
          out DDRD,temp         ; порт PD на введення
          ldi temp,0x02         ;підключення підтягуючого
          out PORTD,temp   ; резистора до PD.2
          ldi temp,0x40          0x40 == 0100_0000 дозвіл
          out GIMSK,temp   ; переривання INT0 (6 біт GIMSK)
          ldi temp,0x00         ;обробка переривання INT0
          out MCUCR,temp   ; по низькому рівню L (табл. 2)
          sei              ;глобальний дозвіл переривань
WAITSTART: sbic PIND,START ; очікування натискання
          rjmp WAITSTART        ;  кнопки START
LOOP: out PORTB,reg_led     ;включення СД
          rcall DELAY           ;затримка
          ser temp         ;виключення
          out PORTB,temp   ; світлодіодів
          brts LEFT        ;перехід, якщо прапор T встановлений
          sbrs reg_led,0   ;пропуск наступної команди,
                           ; якщо 0-й розряд reg_led встановлений
          set              ;T=1
          ror reg_led           ;зсув reg_led вправо на 1 розряд
          rjmp LOOP
LEFT: sbrs reg_led,7   ;пропуск наступної команди,
                           ;якщо 7-й розряд reg_led встановлений
          clt              ;T=0
          rol reg_led           ;зсув reg_led вліво на 1 розряд
```

```
            rjmp LOOP
;***Обробка зовнішнього переривань від кнопки STOP***
STOP_PRESSED:
WAITSTART_2:                ; очікування
            sbic PIND,START ; натискання
            rjmp WAITSTART_2; кнопки START
            reti

;*** Підпрограма затримки ***
DELAY:      ldi r17,0xff    ; Величина затримки підбирається для кожного ПК!
d1:   ldi r18,0x50
d2:   dec r18
            brne d2
            dec r17
            brne d1
            ret
```

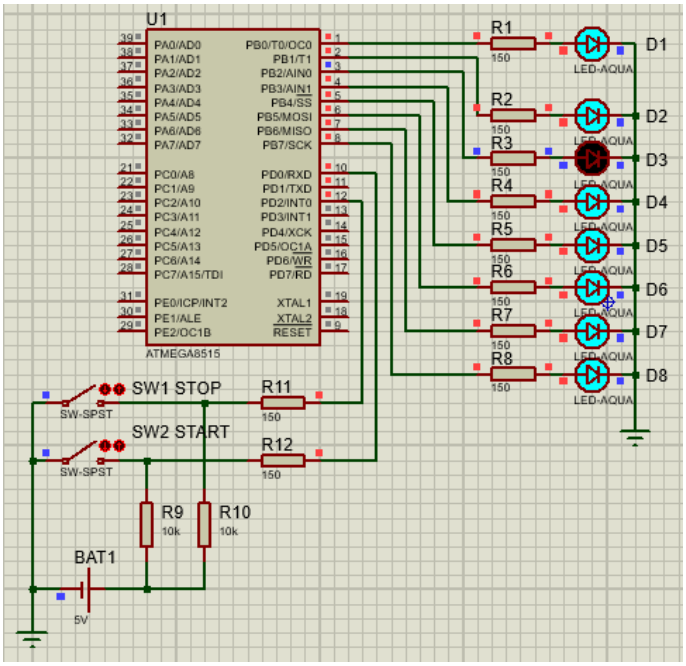


Рисунок 4 – З’єднання кнопок і МК

Програму прикладу 4 можна переробити з використанням зовнішніх переривань. Для цього використовується два переривання – від кнопки (INT0) і від переповнення таймера TIM0_OVR. Для розрізнення стану кнопки, коли МК реагує на наростаючий або спадаючий фронт, використовується регістр прапорів Flag. Програма підраховує кількість натискань кнопкою SW3 і відображає їх кількість у двійковому коді на світлодіодах. Програма виконується в Proteus/Isis, електрична схема з’єднань аналогічна до показаної на рис. 4.

Приклад 5.

```
; ************************************************************
; 5.asm - підрахунок натискань кнопки SW1 з використанням переривання
; ************************************************************
;.include  "m8515def.inc"
.def temp = r16          ;робоча змінна
.def Count_time = r17    ;лічильник затримки
```

```
.def Counter = r18       ;лічильник натискань
.def Flag = r19          ;регістр прапорів, якщо біт 0 встановлений
                         ;то виявлено натискання і перехід до виявлення відпускання
; *** вектор переривань ****
rjmp RESET      ; $000 - Reset Handler
rjmp EXT_INT0   ; $001 - IRQ0 Handler
reti; rjmp EXT_INT1   ; $002 - IRQ1 Handler
reti; rjmp TIM1_CAPT  ; $003 - Timer1 Capture Handler
reti; rjmp TIM1_COMPA ; $004 - Timer1 Compare A Handler
reti; rjmp TIM1_COMPB ; $005 - Timer1 Compare B Handler
reti; rjmp TIM1_OVF   ; $006 - Timer1 Overflow Handler
rjmp TIM0_OVF         ; $007 - Timer0 Overflow Handler
reti; rjmp SPI_STC    ; $008 - SPI Transfer Complete Handler
reti; rjmp USART_RXC  ; $009 - USART RX Complete Handler
reti; rjmp USART_UDRE ; $00a - UDR0 Empty Handler
reti; rjmp USART_TXC  ; $00b  - USART TX Complete Handler
reti; rjmp ANA_COMP   ; $00c - Analog Comparator Handler
reti; rjmp EXT_INT2   ; $00d - IRQ2 Handler
reti; rjmp TIM0_COMP  ; $00e - Timer0 Compare Handler
reti; rjmp EE_RDY     ; $00f - EEPROM Ready Handler
reti; rjmp SPM_RDY    ; $010 - Store Program memory Ready

EXT_INT0:             ; обробник переривання
  clr temp
  out GICR,temp       ; заборона переривань INT0 від кнопки
  ldi temp,0b0100_0000; чистка лог.1 6-розряду регістру прапорів GIFR
  out GIFR,temp ; зміна логічного стану або фронтів INT0 викликає переривання
  sbrs Flag,0   ; перевірка біту 0
  rjmp Push_pin ; якщо 0, то було натискання
  cbr Flag,1    ; інакше було відпускання, очищення біту 0
  inc Counter   ; кнопка відпущена, збільшуємо лічильник
  out PORTB,Counter  ; виведення лічильника в порт
  ldi Count_time,0x9 ; 50 інтервал 0,2 с - розраховується!
  rjmp ent_int;      ; перехід на вихід
Push_pin:            ; було натискання
  sbr Flag,1         ; встановлюємо біт 0
  ldi Count_time,0x9 ; 128 інтервал 0,5 сек розраховується!
ent_int:
  ldi temp,0b0000_0011 ; Запуск Timer0 з вхідною частотою 1:64
  out TCCR0,temp
reti                 ; кінець обробки переривань EXT_INT0

TIM0_OVF:            ; обробник переривань Timer0
  dec Count_time     ; для кожного переривання зменшуємо на 1
  breq end_timer     ; якщо нуль, то на кінець відліку
  reti               ; вихід з переривання
end_timer:
  clr temp           ; зупинка таймера
  out TCCR0,temp
  sbrc Flag,0        ; перевірка біта 0 регістра прапорів
  rjmp Push_tim      ; якщо 1, то було натискання
  ldi temp,(1<<ISC01); інакше встановлюємо пер. EXT_INT0 по спаду
  out MCUCR,temp
  rjmp end_tim       ; на вихід
Push_tim:            ; якщо було натискання
  ldi temp,(1<<ISC01|1<<ISC00) ; встановлюємо переривання по фронту
  out MCUCR,temp
end_tim:
  ldi temp, (1<<INT0) ; дозволяємо переривання INT0
  out GICR,temp
reti                  ; кінець обробки переривань таймера

; *** програма ***
RESET:     ; початкова ініціалізація
ldi temp,low(RAMEND)  ; завантаження вказівника стеку
```

```
out SPL,temp
ldi temp,high(RAMEND)
out SPH,temp

ldi temp,0b0000_0100  ; в 2-й розряд PORTD підтягуючий резистор
out PORTD,temp        ;  після RESET порт налаштований на вхід
ldi temp, 0b1111_1111
out DDRB,temp         ; порт PORTB на виведення
clr Counter
clr Flag
ldi temp,(1<<TOIE0)   ; дозвіл переривання таймера за (TIMSK)
out TIMSK,temp
ldi temp,(1<<ISC01)   ; встановлення переривання INT0 по спаду
out MCUCR,temp
ldi temp,(1<<INT0)    ; дозвіл переривання INT0
out GICR,temp
sei                   ; загальний дозвіл переривання
G_cykl: rjmp G_cykl   ; основний цикл
```

Завдання.

1. В Microchip Studio/Atmel Studio/AVR Studio відкомпілювати програми прикладів. Виконати програми на платі STK500 з цільовим МК та в середовищі Proteus.

2. В програму прикладу 4, добавити зовнішнє переривання INT1 (сигнал на лінії PD3, адрес переривання 002, біт 7 регістра маски GICR, GIFR) від кнопки SW3. Кількість натискань підраховувати і висвічувати на світлодіоди окремо для кнопок SW1 і SW3.

3. Модифікувати програму прикладу 3 так, щоб перемикати світлодіоди при натисканні наступної комбінації кнопок

№	Комбінація кнопок
1	SW1,SW2
2	SW1,SW3
3	SW1,SW7
4	SW2,SW3
5	SW2,SW4
6	SW2,SW5
7	SW2,SW6
8	SW2,SW7
9	SW3,SW4
10	SW3,SW5
11	SW3,SW6
12	SW3,SW7
13	SW4,SW5
14	SW4,SW6
15	SW4,SW7
16	SW5,SW6
17	SW5,SW7
18	SW6,SW7
19	SW1,SW2,SW3
20	SW1,SW2,SW4

Примітка. Номер варіанта завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Текст програми із поясненнями.
4. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми

Питання.

1. Які є типи переривань і від чого вони виникають?
2. Які є способи обробки подій, їх переваги і недоліки?
3. Що таке вектор переривань, пріоритетність, розміщення в пам'яті та призначення?
4. Які виводи портів МК можна використати для зовнішніх переривань?
5. Розказати про регістри SREG, MCUCR, EMCUCR, GICR, GIF які використовуються при обробці зовнішніх переривань.
6. Розказати про регістри TIMSK, TIFR, які використовуються при обробці переривань таймера.
7. Як розрахувати затримку програмного імітатора таймера?
8. Як розрахувати затримку апаратного таймера?

ЛАБОРАТОРНА РОБОТА № 7. РКІ і контролер HD44780

Мета. Отримання практичних навичок роботи з рідкокристалічним індикатором і контролером HD44780.

1. Теоретична частина

1.1. Структура контролера HD4470

Блок схема контролера HD4470 показана на рис. 1.

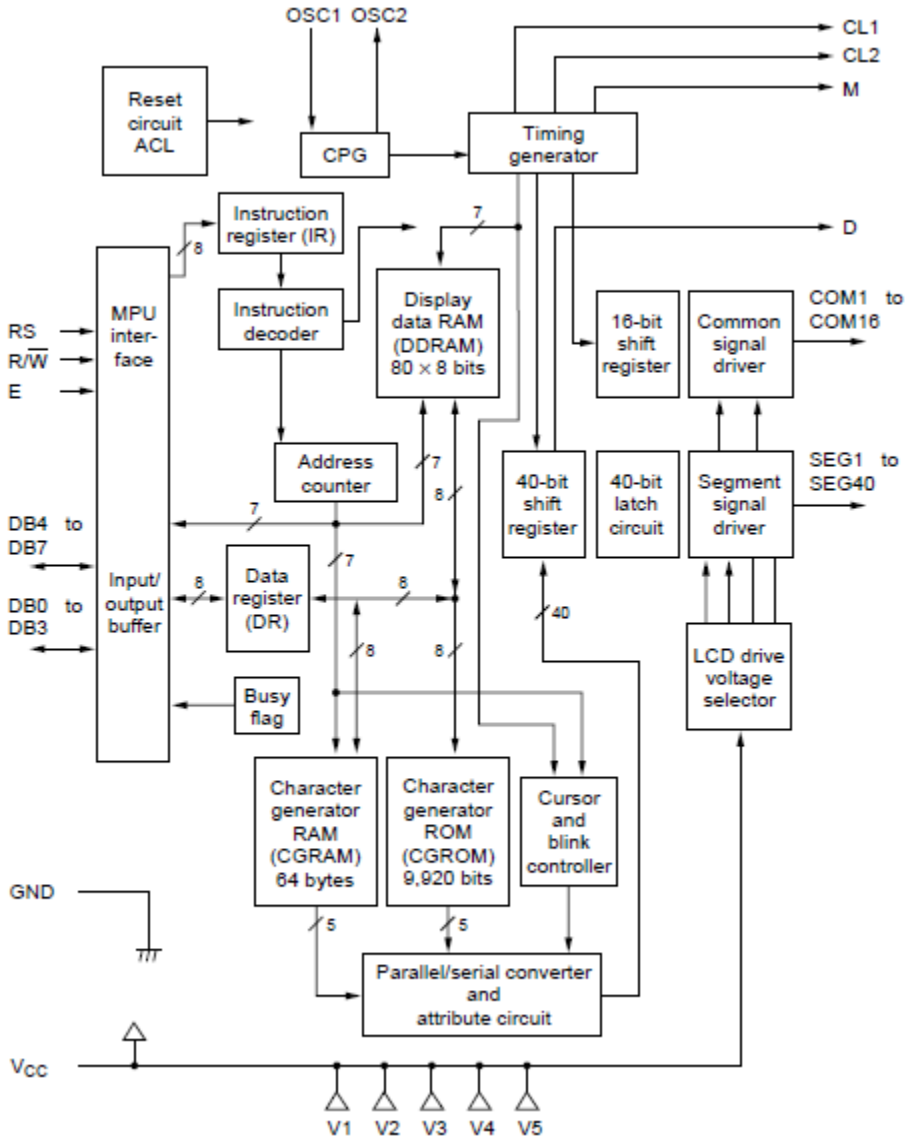


Рис. 1 – Блок схема контролера HD4470

Нумерація і назви виводів контролера HD4470 U (FP-80U) показані на рис. 2.

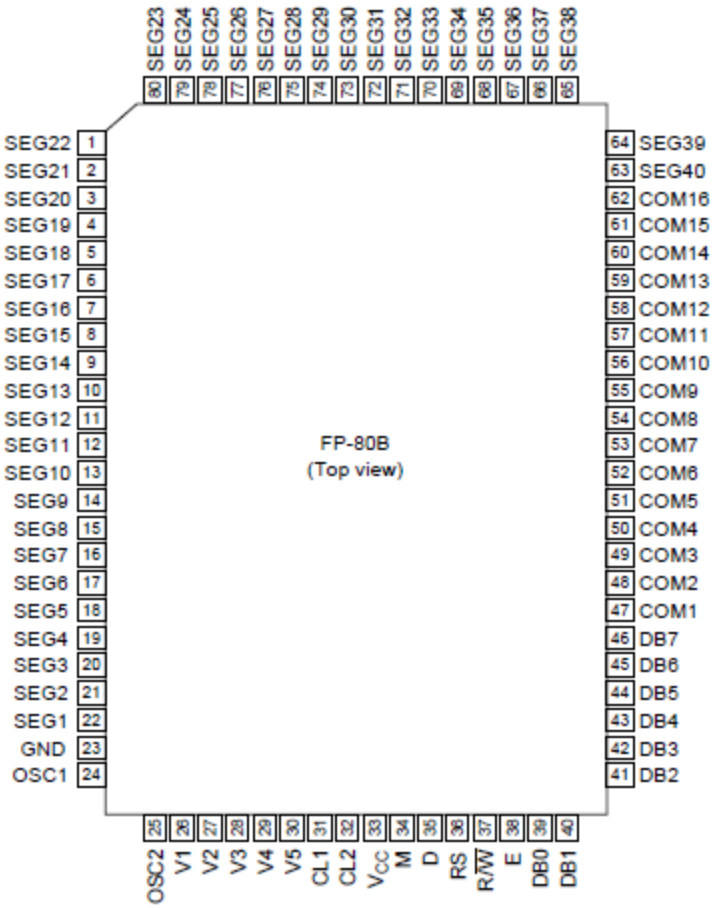


Рис. 2 – Нумерація і назви виводів контролера HD4470 U (FP-80U)

Функції сигналів:

RS (зв’язок з мікроконтролером (МК)) – вибір регістрів: 0 – регістр інструкцій, 1 – регістр даних

R/W’ (зв’язок з МК) – читання(0) / запис (1)

E (зв’язок з МК) – початок читання/запису даних

DB4-DB7 (зв’язок з МК) – старші біти шини даних. DB7 — може використовуватися як прапор зайнятості

DB0-DB3 (зв’язок з МК) – молодші біти шини даних (ці біти не використовуються у 4-бітовому режимі

CL1 (розширення драйвера) – тактування для фіксації послідовних даних D

CL2 (розширення драйвера) – тактування для зсуву послідовних даних D

M (розширення драйвера) – сигнал керування формою сигналів

D (розширення драйвера) – шаблон даних для кожного сегментного сигналу

COM1-COM16 (РКІ) – службові загальні сигнали керування коефіцієнтами тривалості

SEG1-SEG4 (РКІ) – службові сегментні сигнали

V1-V5 – живлення 11 В

Vcc, GND – живлення (2,7 – 5.5 В), GND (0 В)

OSC1, OSC2 – зовнішній резистор для внутрішнього тактового генератора. Зовнішній тактовий генератор під’єднується до OS**C**1.

1.2. Функціональні елементи

Регістри.

Контролер має регістр інструкцій (IR) і регістр даних (DR). Вибір регістрів здійснюється регістром селектором (RS).

IR зберігає інструкції, такі як чистка екрану і зсув курсору, адресну інформацію для відображення даних з DDRAM і символів генератора CGRAM. Регістр IR можна записати тільки з МК.

DR тимчасово зберігає дані, які будуть записуватися у DDRAM або CGRAM і які прочитані з DDRAM або CGRAM. Дані записані з МК у DR автоматично записуються у DDRAM або CGRAM. Коли адреси записуються в IR, дані читаються і зберігаються в DR з DDRAM або CGRAM внутрішніми командами. Передача даних завершується коли МК зчитав дані. Після читання, дані з DDRAM або CGRAM за наступною адресою посилають у DR для наступного зчитування у МК.

Прапор зайнятості (BF).

Коли BF=1 контролер зайнятий і на приймає наступні команди. Коли RS=0 і R/W'=1, значення BF виводиться у DB7. Наступна інструкція може бути записана тільки при BF=0.

Лічильник адрес (AC).

Лічильник адрес назначає адреси DDRAM і CGRAM. Коли адреса інструкції записується в IR, то звідти вона надсилається в AC. Вибір адреси DDRAM чи CGRAM визначається поточною інструкцією.

Після читання/записування DDRAM або CGRAM значення AC автоматично збільшується на 1. Значення AC виводиться в DB0...DB6 (RS=0, R/W'=1, табл. 1).

Таблиця 1 – Вибір регістрів

RS	R/W'	Операція
0	0	IR записується як внутрішня операція
0	1	Читання DB7 і адресація DB0...DB6
1	0	DR записується як внутрішня операція (DR в DDRAM або CGRAM)
1	1	DR читається як внутрішня операція (DDRAM або CGRAM в DR)

Буфер відеопам'яті даних (Display Data RAM - DDRAM).

Буфер має розмір 80×8 біт (80 символів, один символ 8 біт). Цю пам'ять використовують екрани 1 ряд × 8 символів, 2 ряди × 8 символів, 2 ряди × 16 символів. Залежність між адресацією комірок DDRAM і позиціями символів на екрані 2×16 показана на рис. 3.

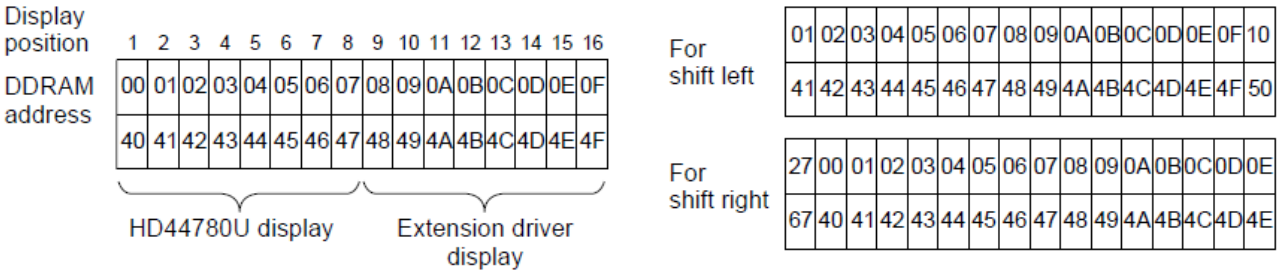


Рис. 3 – Позиції символів на екрані і адресація DDRAM

Пам’ять генератора символів.

Можуть генеруватися шаблони для кодів 8-бітових символів з роздільною здатністю 5×8 або 5×10 крапок (5×8 – 208 шаблонів, 5×10 – 32 шаблони).

Програмно користувач може перезаписати 8 шаблонів символів 5×8 і 4 шаблонів 5×10.

Залежність між кодами символів і шаблонами символів показана на рис. 4.

Рис. 4 – Таблиця символів CGRAM

Для відображення любого символу МК має передати координати позиції і, безпосередньо за ними, саму адресу символу з CGRAM. Великі та малі літери латинського алфавіту, числові знаки, а також більшість знаків пунктуації збігаються в ній з кодами ASCII. Набір символів, розміщених за адресами 0xA0 ... 0xFF, містить національний алфавіт (в даному випадку кирилицю) того регіону, де передбачається його використання. Перші 16 комірок CGRAM мають особливе значення (LCD A162-D фірми Ampire). При бажанні, в них можуть бути записані будь-які символи користувача, яких немає в таблиці (відразу після включення модуля в них знаходиться випадкова інформація).

1.3. Послідовність передачі даних

Послідовність передачі даних в HD44780 показана на рис. 5.

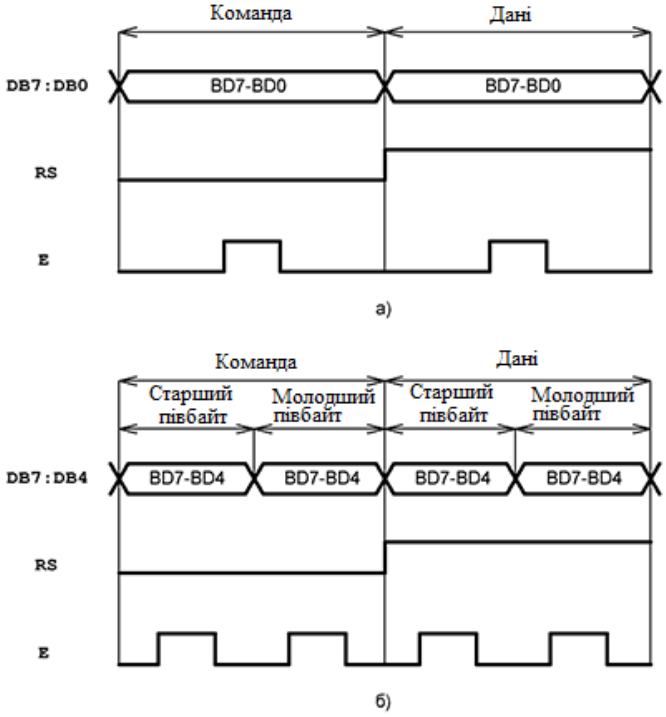


Рис. 5 – Послідовність передачі команд: а) 8-розрядна шина даних/команд;
б) 4-розрядна шина даних/команд

HD44780 взаємодіє з МК через 8-бітову двонаправлену шину команд/даних DB7:DB0. Часова діаграма роботи шини показана на рис. 5. У момент запису інформації в рідкокристалічний індикатор (РКІ) МК виставляє на лініях DB7...DB0 8-розрядний код, після чого формує на виведення E стробуючий імпульс (активний фронт – задній). По закінченню імпульсу повинна бути витримана пауза до початку нової транзакції. Ознакою запису команди/даних є стан лінії RS. При RS = 0 відбувається запис команди, при RS = 1 – даних. Коли необхідно зчитувати дані з індикатора, то виводи порту DB7:DB0 МК налаштовуються на введення. Потім слідує імпульс підтвердження на лінії E і байт даних переписується у внутрішній регістр для подальшої обробки. Напрямок передачі даних визначає рівень на лінії R/W (R/W=1 – читання з індикатора, R/W=0 – запис в індикатор). У реальних застосунках, як правило, немає необхідності в читанні даних з РКІ. Тому виводи R/W завжди з'єднують із загальним провідником. Схема підключення МК до A162-D фірми Ampire наведена на рис. 6.

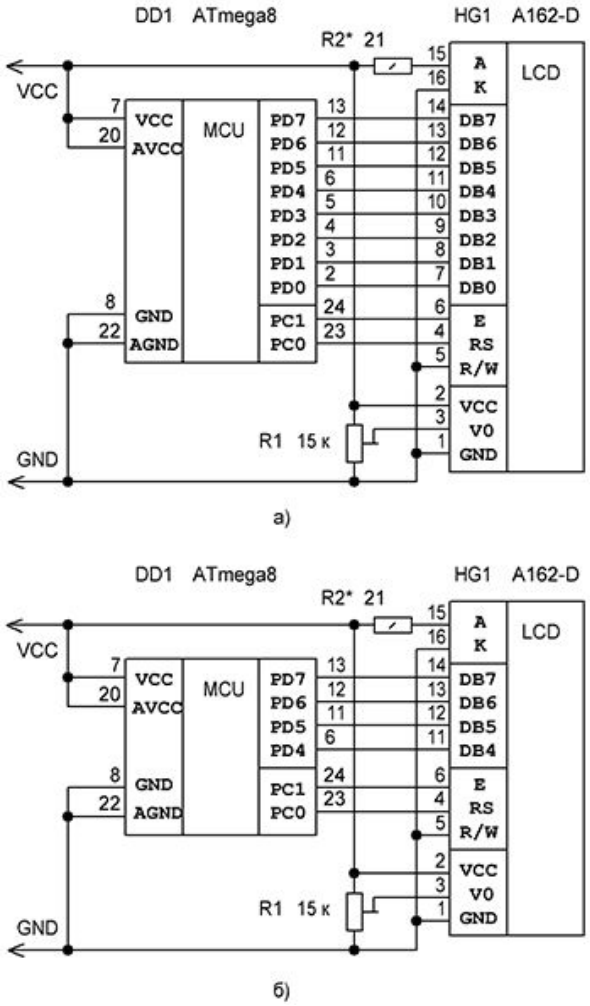


Рис. 6 – Схема підключення РКІ A162-D фірми Ampire до МК:
а) 8-розрядна шина даних/команд;
б) 4-розрядна шина даних/команд

1.4. Команди запису/читання в HD44780

Таблиця 2 – Команди запису

N	Стан ліній, при R/W=0									Команди	Макс. час виконання, мкс
	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
1	0	0	0	0	0	0	0	0	1	Повна чистка екрану і встановлення курсору в нульову позицію.	1600
2	0	0	0	0	0	0	0	1	-	Встановлення курсору в нульову позицію. Встановлення екрану в початковий стан.	1600

3	0	0	0	0	0	0	1	I/D	S	I/D (Increment/Decrement) - напрямок зсуву курсору після запису (I/D=1 - зсув праворуч, I/D=0 – зсув ліворуч). S(Shift) – дозвіл зсуву екрану разом з курсором (S=1 – зсув дозволений, S=0 - зсув заборонений).	40
4	0	0	0	0	0	1	D	C	B	D(Display) – включення екрану (D=1 – включений, D=0 – відключений). C(Cursor) – видимість курсору (C=1 – видимий, C=0 – невидимий). B(Blink) – блимаючий (B=1 – блимає, B=0 – не блимає).	40
5	0	0	0	0	1	S/C	R/L	-	-	S/C(Screen/Cursor) – переміщення екрану/курсора (S/C=1 – переміщається екран, S/C=0 – переміщається курсор). R/L(Right/Left)- напрямок переміщення екрану/курсора (R/L=1 – переміщення вправо, R/L=0 – переміщення вліво).	40
6	0	0	0	1	DL	N	F	-	-	DL(Data Length) – розрядність шини даних (DL=1 – 8 біт, DL=0 – 4 біти). N(Number) – число рядків екрану (N=1 – 2 , N=0 – 1. F(Font) – розмір шрифту (F=1 – 5x10 крапок, F=0 – шрифт 5x7 крапок).	40
7	0	0	1	ADRESS						Установка адреса CGRAM (Character Generator RAM). Після команди мають слідувати дані для запису/читання в/з CGRAM.	40
8	0	1	ADRESS							Встановлення адреси відеобуфера DDRAM . Після команди мають слідувати дані для запису/читання в/з DDRAM.	40
9	1	DATA								Записування даних в DDRAM або CGRAM.	40

Таблиця 3 – Команди читання з HD44780:

N	Стан ліній, при R/W=1									Команди	Макс. час виконання, мкс
	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
1	0	BF	AC							Читання BF (Busy Flag) – прапор завершення операції (BF=1 – завершена, BF=0 – не завершена) и поточного стану внутрішнього лічильника адреси AC.	1
2	1	DATA								Читання даних з DDRAM або CGRAM.	40

У цьому випадку 4-розрядну шину команд/даних формують лінії DB7…DB4 (лінії DB3… DB0 залишаються невикористаними). Швидкість запису знижується в 2 рази. Послідовність передачі даних показана на рис. 5, б. Команди/дані передаються за два такти. Першим іде старший півбайт, другим – молодший. Кожна тетрада має бути зафіксована імпульсом на лінії E.

Керуючі команди запису показані в табл. 2. Запис команди з кодом 0x01 призводить до повної чистки DDRAM і установці вікна екрану і курсора в початкові позиції. Команда 0x02 змушує виконати ті ж самі дії, але при цьому залишає вміст відеопам'яті незмінним. Біти команди під номером 3 задають напрямок зміщення курсору (I/D = 1 – зсув праворуч, I/D = 0 - зсув ліворуч) і дозвіл зсуву екрану (S = 1 – зсув дозволений, S = 0 – зсув заборонений) при введенні чергового символу. Біти команди 4 відповідають за режим відображення курсору (B = 1 – курсор блимає, B = 0 – курсор не блимає; C = 1 – видимий курсор, C = 0 – погашений курсор) і роботу екрану (D = 1 – екран включений, D = 0 – екран відключений). Команду 5 зручно використовувати для реалізації рухомого рядка. За її допомогою можна примусово переміщувати екран або курсор (S/C = 1 – переміщається екран, S/C = 0 – переміщається курсор), в довільному напрямку (R/L = 1 – переміщення праворуч, R/L = 0 – переміщення ліворуч). Вміст DDRAM, в цьому випадку, залишається незмінним. Команда 6 використовується тільки під час початкової ініціалізації модуля. Вона задає тип інтерфейсу (DL = 1 – 8-провідний, DL = 0 – 4-провідний), число рядків дисплея (N = 1 – 2 рядки, N = 0 – 1 рядок) і розмір шрифту (F=1 – шрифт 5x10 крапок (не використовується), F=0 – 5x7 крапок).

Команди 7 і 8 призначені для встановлення поточної адреси в CGRAM і DDRAM, відповідно, і можуть бути використані тільки спільно з командою запису даних 9 (або з командою читання 2, з табл. 3). Після встановлення курсору в пам'яті DDRAM, команда 9 повинна передавати адресу символу (0 ... 0xFF) з таблиці CGRAM для його відображення у відповідній позиції. Комбінація команд 7 і 9 необхідна при програмування символів користувача в CGRAM за адресами 0 ... 0x0F. Для запису кожного символу потрібно 8 байт пам'яті МК. Корисну інформацію нестимуть в собі тільки 5 молодших розрядів, відповідних 5-ти стовпцям матриці (рис. 7). Логічній одиниці відповідає видима крапка на дисплеї. Після встановлення адреси в ОЗП знакогенератора, повинен слідувати 8-байтовий блок даних. Можливий запис декількох символів поспіль. Так, наприклад, щоб запрограмувати всі 16 символів, потрібно передати команду 0x40 (встановити нульову адресу в CGRAM), а за нею 16*8 = 128 байт даних.

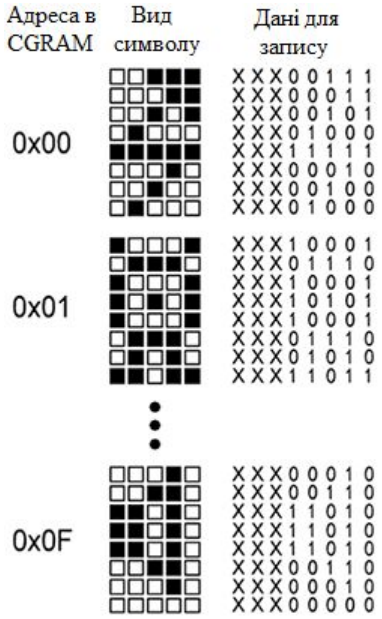


Рис. 7 – Символи користувача

Команди читання з індикатора показані в табл. 3. За допомогою першої команди може бути прочитано поточний зміст лічильника адреси AC в DDRAM і стан прапора завершення операції BF (при BF = 1 операція читання/запису завершена). Команда 2 повинна слідувати після команди записи 7 або 8 з табл. 2 і дозволяє зчитувати символи розміщені в CGRAM або DDRAM. Як зазначалося вище, команди читання не мають ніякої практичної цінності. Інтерес може являти тільки прапор BF. Однак набагато зручніше програмно формувати затримки часу, що гарантують завершення операцій читання/запису, ніж постійно опитувати стан прапора закінчення операції. Необхідність використання лінії R/W при цьому також відпадає.

2. Практична частина

2.1. Приклади алгоритмів для виконання завдань на РКІ HD44780

Завдання 1.

1. Включити дисплей.
2. Очистити вміст.
3. Зсунути курсор на одну позицію.
4. Записати на місце, куди вказує курсор, «1».

Рішення (послідовність команд).

Насамперед ініціалізація екрану без чого більша частина РКІ на HD44780 не буде працювати. Деякі моделі мають значення за замовчуванням (шина 8 біт, курсор в 0). Але краще це зробити явно.

1. **001**11000 Шина 8 біт, 2 рядки.
2. **00000001** Очищення екрану.
3. **00000**110 Інкремент адреси. Екран не рухається.

1. **00001**100 Включення екрану (D = 1).

2. **00000001** Очищення екрану. Вказівник встановлений на DDRAM.
3. **0001**0100 Зсув курсору (S/C = 0) праворуч (R/L = 1).
4. 00110001 Запис даних (RS = 1) код «1» 0x31.

Жирним шрифтом позначені ідентифікатори команд.

Завдання 2.

Створити свій символ. З кодом 01 і вивести його на екран.

Вважаємо, що екран ініціалізований і готовий до прийому даних.

Рішення.

1. **01**001000 Вибір в CGRAM адреси 0x08 – якраз початок другого символу (один символ 8 байтів)
2. 00000001 Пересилання 8 байт даних, (RS = 1).
3. 00000010 Символ блискавки
4. 00000100
5. 00001000
6. 000**11111** Старші три біти не використовуються.
7. 00000010
8. 00000100
9. 00001000 Останній байт даних.
10. **1**0000000 Команда перемикання адреси в DDRAM і вказівник на адресу **0000000** – перший символ в першому рядку.
11. 00000001 Знову дані (RS = 1), код 01 – сюди записаний символ блискавки і висвічується на екрані.

Алгоритм читання/запису в РКІ HD44780.

Ініціалізація портів.

1. RS, RW, E Режим виходу.
2. DB7..DB0 Режим входу.

Очікування готовності, читання прапора зайнятості.

1. Порт даних на вхід з підтягуючими резисторами (DDR = 0, PORT = 1).
2. RS = 0 (команда)
3. RW = 1 (читання)
4. E = 1 (стан готовності)
5. Пауза (14 тактів процесора на 8МГц)
6. E = 0
7. Читання з порту. Якщо біт 7 (Busy flag) встановлений, то повторити всі кроки, поки біт не скинеться.

Запис команди.

1. Очікування готовності.
2. RS = 0 (команда).
3. RW = 0 (запис).
4. E = 1 (стан готовності).
5. Порт на вихід.

6. Вивести в порт код команди.
7. Пауза.
8. E = 0
9. Порт на вхід.

Запис даних.

1. Очікування готовності.
2. RS = 1 (Дані).
3. RW = 0 (запис).
4. E = 1 (стан готовності).
5. Порт на вихід.
6. Вивести в порт код команди.
7. Пауза.
8. E = 0 .
9. Порт на вхід.

Читання команди.

1. Очікування готовності.
2. Порт даних на вхід з підтягуючими резисторами (DDR = 0, PORT = 1).
3. RS = 0 (команда).
4. RW = 1 (читання).
5. E = 1 (Стан готовності. В цей момент дані з РКІ поступають на шину).
6. Пауза
7. Читання даних з порту.
8. E = 0.

Читання даних.

1. Очікування готовності.
2. Порт даних на вхід з підтягуючими резисторами (DDR = 0, PORT = 1)..
3. RS = 1 (Дані)
4. RW = 1 (читання).
5. E = 1 (Стан готовності. В цей момент дані з РКІ поступають на шину).
6. Пауза
7. Читання даних з порту.

8. E = 0.

2.2. Програма для роботи з РКІ HD44780

Набір підпрограм для роботи з символьним РКІ HD44780 наведено нижче. Підпрограми `write_com`, `write_dat` записують команди і дані відповідно. Підпрограма `show_char` виводить символ на екран дисплея; `show_string` переписує рядок, що зберігається у FLASH-пам'яті програм, в DDRAM РКІ. Обидві підпрограми як параметри приймають початкові координати запису – рядок і стовпець. У `show_string`, крім цього необхідно передати ще й покажчик на рядок в регістрі ZH: ZL. Окремо слід сказати про підпрограму ініціалізації `hd44780_init`, яка повинна бути викликана після подачі напруги живлення на модуль, так як в ній можуть

виникнути деякі проблеми. Послідовність команд в ході цієї процедури може мати невеликі відмінності у індикаторів різних типів. Тому необхідно звертатися до технічної документації на конкретну модель. Неправильна ініціалізація, зазвичай, призводить до повної непрацездатності справного індикатора.

Програма роботи з РКІ HD44780.

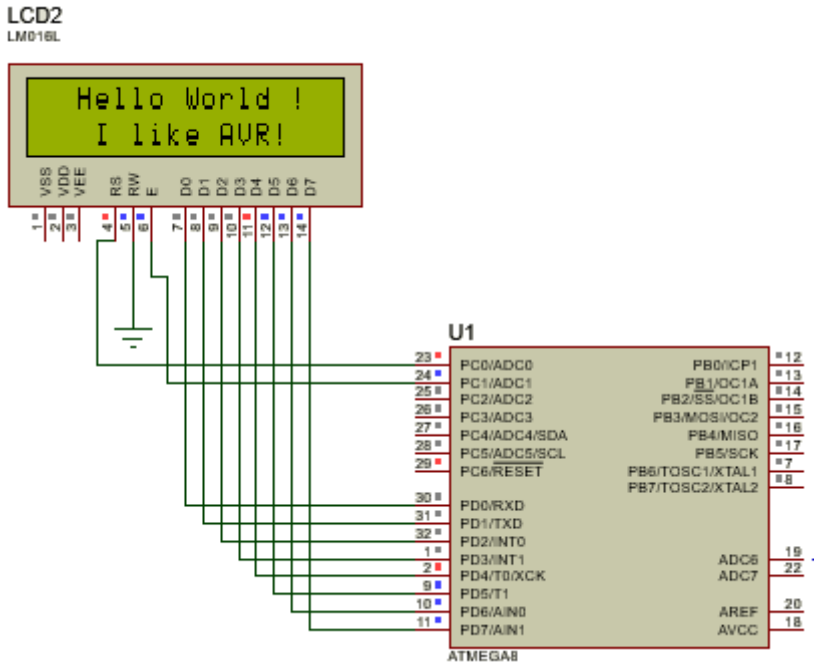
```
     ; AssemblerApplication1.asm
; Created: 4/10/2021 12:53:09 PM
; Author : user
; Replace with your application code
     .def data = R16  ;регістр для передачі команд і даних
     .def row  = R17  ;регістр з номером рядка
     .def col  = R18  ;регістр з номером позиції на індикаторі
     .def temp = R19  ;регістр для проміжкових операцій

     .equ DELAY = 500 ;затримка часу на частоті 1 МГЦ Atmega 8
     .equ RS = PC0
     .equ EN = PC1

     ldi   temp,high(RAMEND) ;ініціалізація стеку
     out   SPH,temp
     ldi   temp,low(RAMEND)
     out   SPL,temp
    .
     clr   temp            ; обнулення регістрів PORTD, PORTC
     out   PORTD,temp
     out   PORTC,temp
ldi temp,0b11110000;для шини 4 біти налаштовуємо PD7…PD4 на вивід
;ldi temp,0b11111111;для шини 8 біт налаштовуємо PD7…PD0 на вивід
     out   DDRD,temp
     ldi   temp,0b00000011 ;налаштування лінії PC1,PC0 на вивід
     out   DDRC,temp
     rcall hd44780_init;ініціалізація модуля перед початком роботи
     ldi   row,1        ;надпис для виведення "Hello World !"
     ldi   col,3        ;в рядку 1 з позиції 3
     ldi   ZH,high(2*first_string)
     ldi   ZL,low(2*first_string)
     rcall show_string
     ldi   row,2        ;виведення надпису "I like AVR”
     ldi   col,4        ;в рядку 2 з позиції 4
     ldi   ZH,high(2*second_string)
     ldi   ZL,low(2*second_string)
     rcall show_string
    .
first_string:
; рядок з 13 символів "Hello World !"
 .db   "Hello World !",0x20,0
second_string:
; рядок з 11 символів "Я люблю AVR”
; .db  0x20,0xB1,0x20,0xBB,0xC6,0xB2,0xBB,0xC6
; .db  0x20,0x41,0x56,0x52,0x20,0x20,0x20,0x20,0
  .db  "I like AVR",0x20,0
; Набір підпрограм для роботи з HD44780(16 символів х 2 рядки)
; R16 – регістр для передачі команд і даних
; R17,R18 – лічильники циклів, регістри для задання номеру рядка
;   і позиції у відеопам'яті індикатора
; XH:XL – використовується як 2-байтовий регістр для затримки часу
; ZH:ZL – вказівник на рядок символів, що зберігається в FLASH-пам’яті програм
; DELAY – постійна для затримки часу (500…10000)

;   Допоміжні підпрограми запису байта при використанні 4-провідного інтерфейсу
; Через R16 в підпрограму передається байт для запису
```

```
write_byte:
     push  R16            ;збереження регістру з байтом даних
     andi  R16,0xF0       ;виділення старшого півбайту в регістрі
     in    R17,PORTD      ;копіювання вмісту порту у тимчасовий
     andi  R17,0x0F       ;регістр для модифікації
     or R16,R17           ;залишення незмінним стану молодших 4-х ліній
     sbi   PORTC,EN       ;встановлення на лінії EN лог.1
     out   PORTD,R16      ;виведення нового значення в порт
     ldi   XH,high(DELAY) ;формування затримки часу
     ldi   XL,low(DELAY)
     rcall pause
     cbi   PORTC,EN       ;встановлення на лінії EN лог.0
     pop   R16            ;відновлення регістру з байтом даних
     swap  R16            ;обмін півбайтів для виділення молодших 4-х біт
     andi  R16,0xF0       ;виведення старшого півбайту в регістрі
     in    R17,PORTD      ;копіювання вмісту порту у тимчасовий
     andi  R17,0x0F       ;регістр для модифікації
     or R16,R17           ;залишення незмінним стану молодших 4-х ліній
     sbi   PORTC,EN       ;встановлення на лінії EN лог.1
     out   PORTD,R16      ;виведення нового значення в порт
     ldi   XH,high(DELAY) ;формування затримки часу
     ldi   XL,low(DELAY)
     rcall pause
     cbi   PORTC,EN       ;встановлення на лінії EN лог.0
     ldi   XH,high(DELAY) ;формування затримки часу
     ldi   XL,low(DELAY)
     rcall pause
     ret

;   Допоміжна підпрограма запису байта при використанні
;                 8-провідного інтерфейсу
; Через R16 в підпрограму передається байт для запису

;write_byte:
;    sbi   PORTC,EN       ;встановлення на лінії EN лог.1
;    out   PORTD,R16      ;виведення байту інформації в порт
;    ldi   XH,high(DELAY) ;формування часу затримки
;    ldi   XL,low(DELAY)
;    rcall pause
;    cbi   PORTC,EN       ;встановлення на лінії EN лог.0
;    ldi   XH,high(DELAY) ;формування затримки часу
;    ldi   XL,low(DELAY)
;    rcall pause
;    ret

;    Допоміжна підпрограма затримки часу
pause:
     sbiw  XH:XL,1
     brne  pause
     ret

;            Підпрограма запису команди
;  Через R16 передається код команди перед викликом підпрограми

write_com:
     cbi   PORTC,RS    ;встановлення на на лінії RS лог.0
     rjmp  write_byte  ;запис команди
     ret

;             Підпрограма запису даних
;  Через R16 передається байт даних перед викликом підпрограми

write_dat:
     sbi   PORTC,RS    ;встановлення на лінії RS лог.1
     rjmp  write_byte  ;запис даних
```

```
     ret

;               Підпрограма запису символу
;   Через R16 передається код символу (0…255), в R17 номер рядка
; (1,2), в R18 номер символу в рядку (1…40) при вході в підпрограму

show_char:
     push  R16       ;збереження регістру з символом
     subi  R18,-0x7F ;якщо запис іде в рядок 1, для отримання
     sbrc  R17,1     ;адреси в DDRAM то до номеру символу в рядку
     subi  R18,-0x40 ;додається 0x7F, якщо в рядок 2,
     mov   R16,R18   ;то додається 0xBF
     rcall write_com ;задання адреси символу в відеопам'яті
     pop   R16       ;відновлення регістру з символом
     rcall write_dat ;запис символу за поточною адресою
     ret

;     Підпрограма запису рядка символів з FLASH-пам'яті
;  У вказівник Z передається адреса рядка з FLASH-пам'яті (рядок
; має закінчуватися нулем), в R17 номер рядка (1,2), в R18
; початковий номер позиції, з якої має починатися рядок
; (1…40) при вході в підпрограму

show_string:
     lpm   R16,Z+  ;завантаження коду символу з рядка
     tst   R16     ;якщо код 0 (NUL), то рядок закінчився
     brne  ss1
     ret           ;і вихід з підпрограми
ss1: push  R17     ;збереження регістру з номером рядка
     push  R18     ;збереження регістру з початковим номером позиції
     rcall show_char ;записування символу
     pop  R18;відновлення регістру з початковим номером позиції
     pop   R17 ;відновлення регістру з номером рядка
     inc   R18 ;збільшення на 1 номера позиції
     rjmp  show_string

;              Підпрограма ініціалізації

hd44780_init:
    ldi   XH,high(5*DELAY) ;формування затримки часу
    ldi   XL,low(5*DELAY)
    rcall pause
    ldi   R16,0x33   ;запис необхідної для ініціалізації
    rcall write_com  ;порожньої команди
    ldi   XH,high(5*DELAY) ;формування затримки часу
    ldi   XL,low(5*DELAY)
    rcall pause
    ldi   R16,0x32   ;запис необхідної для ініціалізації
    rcall write_com  ;порожньої команди
    ldi   R16,0x28 ;вибір розрядності шини 4 біта і 2 рядки
;   ldi   R16,0x38;вибір розрядності шини 8 біт і 2 рядки
    rcall write_com  ;на екрані
    ldi   XH,high(5*DELAY) ;формування затримки часу
    ldi   XL,low(5*DELAY)
    rcall pause
    ldi   R16,0x08   ;запис команди виключення екрану
    rcall write_com
    ldi   R16,0x01   ;запис команди початкового скидання
    rcall write_com
    ldi   XH,high(5*DELAY) ;формування затримки часу
    ldi   XL,low(5*DELAY)
    rcall pause
    ldi   R16,0x06   ;запис команди, яка задає напрям
    rcall write_com  ;зсув курсора праворуч
    ldi   R16,0x0C   ;запис команди включення екрану
```

```
rcall write_com
ret
```

Результат виконання програми:



Завдання.

Вивести на екран РКІ HD44780 текстовий рядок.

N	Тип екрану	Роздільна здатність	Шина даних	Тип тексту	Текст
1	1 ряд × 8 сим	5 × 8	4 біт	блимаючий	Hello World
2	2 ряд × 8 сим	5 × 10	8 біт	звичайний	Hello I am Ivan
3	1 ряд × 16 сим	5 × 8	4 біт	блимаючий	Hello from Petro
4	2 ряд × 16 сим	5 × 10	8 біт	звичайний	Hello I am Stepan. I like Atmel
5	1 ряд × 8 сим	5 × 8	4 біт	блимаючий	Hello from Stepan
6	2 ряд × 8 сим	5 × 10	8 біт	звичайний	Hello from Mykola I am student
7	1 ряд × 16 сим	5 × 8	4 біт	блимаючий	Hello from Dmytro
8	2 ряд × 16 сим	5 × 10	8 біт	звичайний	Hello from Vasyl I have home tasks

Примітка. Номер варіанту вибирається згідно номера студента в журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Текст програми із поясненнями.

4. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми

Питання.

1. Які є способи підключення РКІ з контролером HD44780.
2. Які типи пам’яті має контролер HD44780.
3. Призначення і особливості пам’яті DDRAM.
4. Призначення і особливості пам’яті CGROM.
5. Призначення і особливості пам’яті CGRAM.
6. Система команд контролера HD44780.
7. Алгоритм виведення символу на екран РКІ з контролером HD44780.
8. Алгоритм запису символів користувача в CGRAM.

ЛАБОРАТОРНА РОБОТА № 8. Програмування таймерів/лічильників

Мета роботи – вивчення режимів роботи таймерів/лічильників та їх практичного програмування.

1. Призначення таймерів лічильників

Мікроконтролери (МК) AVR в залежності від марки мають в своєму складі від одного до трьох таймерів/лічильників загального призначення – T0, T1 і T2. Основне призначення таймера-лічильника відраховувати часові інтервали. Крім цього вони можуть виконувати додаткові функції — підраховувати тривалість і кількість вхідних імпульсів, формувати ШІМ сигнали.

Таймери/лічильники використовують наступні виводи МК:

OC0/T0 (PB0) – вихід схема співпадіння таймера/лічильника T0 / таймер-лічильник T0;

T1 (PB1) – таймер-лічильник T1;

OC1A (PD5) – вихід схема A співпадіння таймера/лічильника T1.

ICP (PE0) – вхідний регістр захоплення зовнішньої події.

OC1B (PE2) – вихід схема B співпадіння таймера/лічильника T1.

Перший таймер (8-розрядний T0), який є у всіх моделях, може використовуватися для формування і вимірювання часових інтервалів або як лічильник зовнішніх подій, а в моделі ATmega8515 ще і для порівняння із заданим значенням, генерації ШІМ сигналу. При переповненні лічильного регістра таймера генерується запит на переривання. Два інших таймери (16-розрядний T1 и 8-розрядний T2) крім названих мають ще додаткові функції. Обидва таймери можуть генерувати запит на переривання не тільки при переповненні лічильного регістра, але і від ряду інших подій. Вони можуть також використовуватися як широтно-імпульсні модулятори. Крім того, таймер T2 може працювати в асинхронному (відносно тактового сигналу МК) режимі.

Таймери/лічильники працюють незалежно від інших блоків МК, тому паралельно можуть виконуватися інші операції. При організації затримок з декрементом регістрів, інші операції в МК виконати не можна.

Виводи МК ATmega8515, які відносяться до таймерів/лічильників показані в табл. 1.

Таблиця 1 – Виводи таймерів-лічильників

Назва	ATmega8515	STK500	Описання
T0	PB0	PB0	зовнішній сигнал таймера T0
OC0	PB0	PB0	співпадіння схеми порівняння таймера T0
T1	PB1	PB1	зовнішній сигнал таймера T1
ICP	ICP/PE0	PE0	захоплення таймера T1
OC1A	PD5	PD5	співпадіння схеми A порівняння таймера T1
OC1B	OC1B/PE2	PE2	співпадіння схеми B порівняння таймера T1

При використанні портів введення/виведення необхідно сконфігурувати виводи у відповідності з їх функціональним призначенням (вхід або вихід). У всіх МК AVR є також сторожовий таймер, який використовується для запобігання зацикленню програм.

2. Таймер/лічильник Т0 МК ATmega8515

Таймер/лічильник Т0 МК ATmega8515 у своєму складі має:
- базовий лічильник TCNT0, регістр порівняння OCR0, регістр керування TCCR0.

Крім цього є ще три регістри, які відносяться до всіх таймерів-лічильників:
- конфігураційний регістр масок переривань TIMSK;
- статусний регістр запитів переривань TIFR;
- регістр спеціальних функцій SFIOR.

Таймер/лічильник Т0 може працювати в наступних режимах:
- одноканальний лічильник;
- очистка таймера при співпадінні із значенням в OCR0;
- фазокоректний ШІМ;
- генератор частоти;
- лічильник зовнішніх подій;
- джерело переривань при переповненні і співпадінні при порівнянні (TOV0 і OCF0).

TCNT0 це 8-ми розрядний рахунковий регістр, табл. 2. Коли таймер працює, по кожному імпульсу тактового сигналу значення TCNT0 змінюється на одиницю. Залежно від режиму роботи таймера, значення може збільшуватися або зменшуватися. Регістр TCNT0 можна як читати, так і записувати. Останнє використовується коли потрібно задати його початкове значення. Коли таймер TCNT0 працює, змінювати його вміст не рекомендується, так як це блокує схему порівняння на один такт.

Таблиця 2 – Формат регістра TCNT0

Розряд	7	6	5	4	3	2	1	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

TCCR0 – конфігураційний регістр таймера-лічильника Т0, табл. 3. Він визначає джерело тактування таймера, коефіцієнт масштабування подільника, режим роботи таймера-лічильника Т0 і поведінку виведення в ОС0. Біти CS02, CS01, CS00 (Clock Select) – визначають запуск і зупинка лічильника, джерело тактової частоти для таймера Т0 і задають коефіцієнт попереднього подільника, табл. 4.

Таблиця 3 – Формат регістра TCCR0

Розряд	7	6	5	4	3	2	1	0
Ім'я	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

FOC0 – включити вихід порівняння.
WGM00, WGM01 – задають режим роботи генератора форм сигналів.
COM01, COM00 – режим виводу порівняння.

Режим	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter mode	TOP	Зміна OCR0 при співпадінні	Встановлення TOV0 прапора при
0	0	0	Normal	0xFF	Безпосередня	MAX
1	0	1	PWM, phase correct	0xFF	TOP	BOTTON

2	1	0	CTC	OCR0	Безпосередня	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Режим порівняння, non-PWM режим

COM01	COM00	Описання
0	0	Нормальне функціонування порту, OC0 відключено
0	1	Перемкнути OC0 при співпадінні
1	0	Очистити OC0 при співпадінні
1	1	Встановити OC0 при співпадінні

Режим порівняння, Fast PWM режим

COM01	COM00	Описання
0	0	Нормальне функціонування порту, OC0 відключено
0	1	Резерв
1	0	Очистити OC0 при співпадінні, встановити OC0 при TOP
1	1	Встановити OC0 при співпадінні, очистити OC0 при TOP

Режим порівняння, Phase correct PWM режим

COM01	COM00	Описання
0	0	Нормальне функціонування порту, OC0 відключено
0	1	Резерв
1	0	Очистити OC0 при співпадінні при рахунку вверх. Встановити OC0 при співпадінні при рахунку вниз
1	1	Встановити OC0 при співпадінні при рахунку вверх. Очистити OC0 при співпадінні при рахунку вниз

Таблиця 4 – Налаштування тактового сигналу для таймера/лічильника T0

CS02	CS01	CS00	Джерело тактового сигналу
0	0	0	Таймер/лічильник зупинений
0	0	1	Тактова частота CLK (без масштабування)
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	Вивід T0, інкремент лічильника по *спадаючому* фронту імпульсів
1	1	1	Вивід T0, інкремент лічильника по *наростаючому* фронту імпульсів

При використанні таймера/лічильника в режимі рахування зовнішніх подій необхідно врахувати, що сигнал, присутній на виводі T0, синхронізується частотою тактового генератора МК (стан виводу T0 зчитується по наростаючому фронту внутрішнього тактового сигналу). В зв’язку з цим, для забезпечення коректної роботи таймера від зовнішнього сигналу, проміжок

часу між сусідніми імпульсами повинен бути більшим періоду тактового сигналу МК. Інкремент вмісту таймера/лічильника при роботі в режимі підрахунку зовнішніх подій здійснюється навіть у тому випадку, якщо вивід T0 сконфігуровано як вихід. Ця особливість дає користувачу можливість програмно керувати процесом рахування.

OCR0 – 8-розрядний регістр порівняння, табл. 5. Його значення постійно порівнюється з лічильним регістром TCNT0, і в разі збігу таймер може виконувати якісь дії – викликати переривання, змінювати стан виводу OC0 залежності від режиму роботи.

Таблиця 5 – Формат регістра OCR0

Розряд	7	6	5	4	3	2	1	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

3. Таймер/лічильник T1

Таймер/лічильник T1має розширені функціональні можливості і особливості:
- 16-розрядний таймер лічильник TCNT1;
- два незалежних регістри порівняння OCR1A, OCR1B;
- вхідний регістр захоплення ICR1;
- очистка таймера при співпадінні (авто завантаження);
- фазо-коректний ШІМ без збоїв;
- змінний період ШІМ;
- генератор частоти;
- чотири незалежних джерела переривань (TOV1, OCF1A, OCF1B, ICF1).

Структура таймера-лічильника показана T1 на рис. 1.

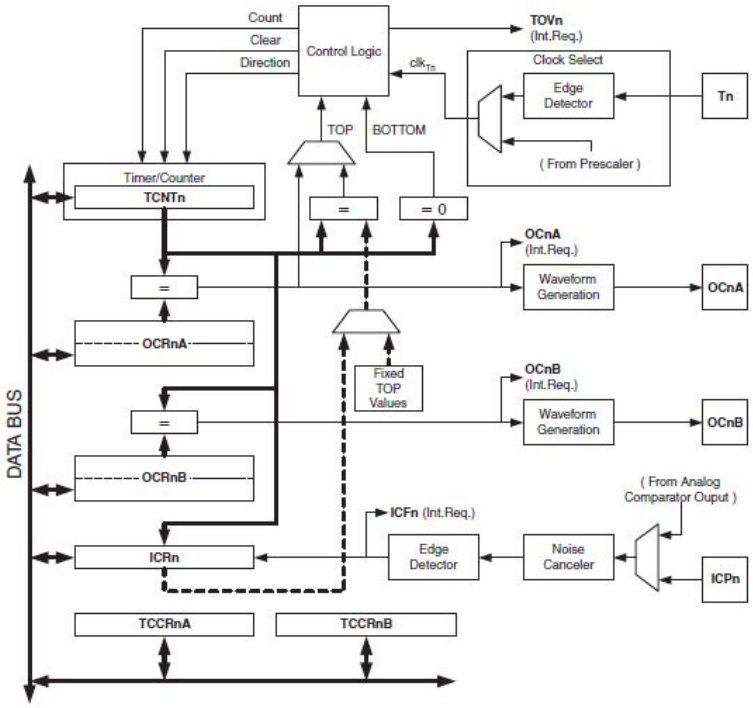


Рисунок 1 – Структура таймера-лічильника T1

3.1. Блок схема таймера лічильника T1

Таймер/лічильник T1 тактується внутрішньо від попереднього дільника. Значення лічильника змінюється від 0 (0x0000) до 65535 (0xFFFF). Під час роботи значення лічильника порівнюється із значення регістрів порівняння OCR1A, OCR1B. Результат порівняння може використовуватися для генерації ШІМ сигналу або змінної частоти на виходах OC1A/B. Подія співпадіння встановлює прапор OCF1A/B, який може використовуватися для генерації переривання.

Для тактування таймера/лічильника може бути використано внутрішнє або зовнішнє джерело. Таймер/лічильник TCCN1 програмується і є двонаправленим. Блок схема лічильника і показана на рис. 2.

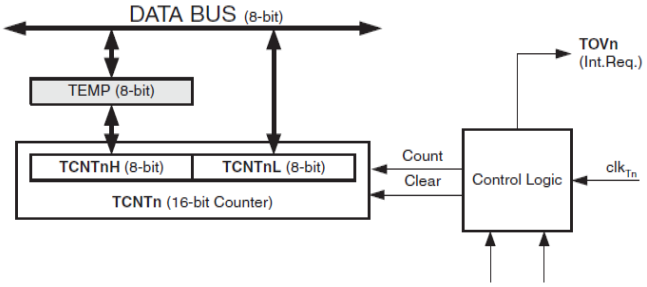


Рис. 2 – Блок схема лічильника

3.2. Блок порівняння лічильника T1

16-розрядний компаратор постійно порівнює значення лічильника TCNT1 із значенням регістрів порівняння OCR1A/OCR1B. При дозволі на порівняння (OCIE=1) і при співпадінні значень встановлюється прапор OCF1A/OCF1B і генерується переривання. Це переривання використовує генератор форм сигналів для генерування виходу згідно встановленого режиму WGM1 і режиму порівняння COM1x1. Блок схема режиму порівняння показана на рис. 3.

Регістри TCNT1, OCR1A, OCR1B є 16-розрядними (табл. 6, 7, 8), тому для читання/запису використовується тимчасовий 8-розрядний регістр TEMP.

Таблиця 6 – Формат регістра TCNT1

Розряд	15	14	13	12	11	10	9	8
	7	6	5	4	3	2	1	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Таблиця 7 – Формат регістра OCR1A

Розряд	15	14	13	12	11	10	9	8
	7	6	5	4	3	2	1	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Таблиця 8 – Формат регістра OCR1B

Розряд	15	14	13	12	11	10	9	8
	7	6	5	4	3	2	1	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Схема блоку порівняння показана на рис. 3.

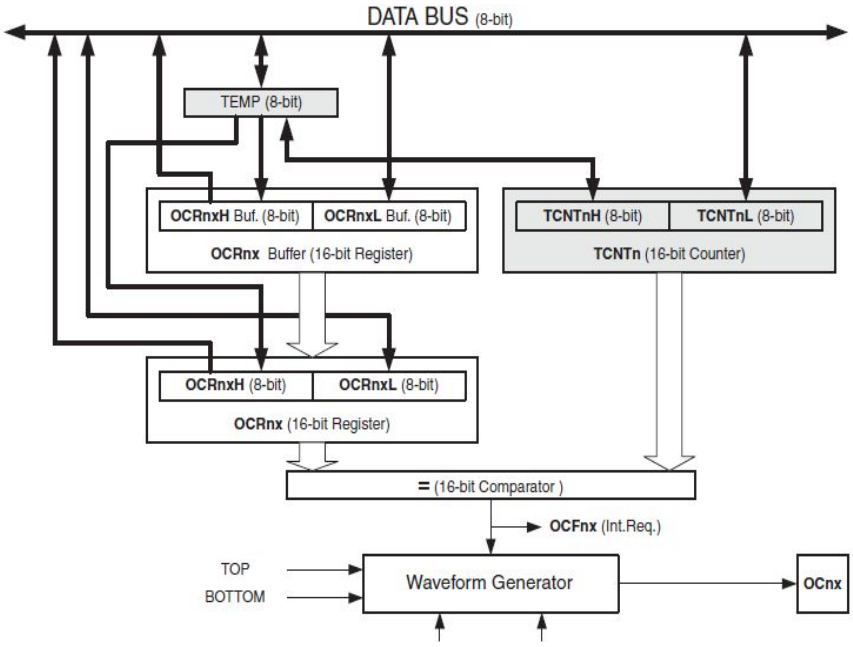


Рис. 3 – Блок схема блоку порівняння

3.3. Конфігураційний регістр TCCR1A

TCCR1A – конфігураційний регістр таймера-лічильника T1, табл. 9. Відповідність між розрядами і режимами роботи таймера/лічильника показана в табл. 9-13. Решта розрядів регістра доступні тільки для читання і містять 0.

Таблиця 9 – Формат регістра TCCR1A

Розряд	7	6	5	4	3	2	1	0
Ім'я	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Біт 7:6 – COM1A1:0: Режим порівняння для каналу A
Біт 5:4 – COM1B1:0: Режим порівняння для каналу B
Біт 3 – FOC1A: Включити вихід порівняння для каналу A
Біт 2 – FOC1B: Включити вихід порівняння для каналу B
Біт 1:0 – WGM1 1:0: Режим роботи генератора форми сигналів

Таблиця 10 – Режим порівняння, non-PWM режим

COM1A1/ COM1B1	COM1A0/ COM1A1	Описання
0	0	Нормальне функціонування порту, OC1A, OC1B відключено
0	1	Перемкнути OC1A/OC1B при співпадінні
1	0	Очистити OC1A/OC1B при співпадінні
1	1	Встановити OC1A/OC1B при співпадінні

Таблиця 11 – Режим порівняння, Fast PWM режим

COM1A1/ COM1B1	COM1A0/ COM1B0	Описання
0	0	Нормальне функціонування порту, OC1A/OC1B відключено
0	1	WGM1 3:0=15. Переключити OC1A при співпадінні, OC1B від'єднати (Нормальний режим портів). Для всіх інших дані WGM1, функціонування портів нормальне, OC1A/OC1B від'єднані.
1	0	Очистити OC1A/OC1B при співпадінні, встановити OC1A/OC1B при TOP
1	1	Встановити OC1A/OC1B при співпадінні, очистити OC1A/OC1B при TOP

Таблиця 12 – Фазо-коректний PWM, фазо- і частотно-коректний PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Описання
0	0	Нормальне функціонування порту, OC1A/OC1B відключено
0	1	WGM1 3:0=9 або 11. Переключити OC1A при співпадінні, OC1B від'єднати (Нормальний режим портів). Для всіх інших дані WGM1, функціонування портів нормальне, OC1A/OC1B від'єднані.
1	0	Очистити OC1A/OC1B при співпадінні при рахунку вверх. Встановити OC1A/OC1B при співпадінні при рахунку вниз
1	1	Встановити OC1A/OC1B при співпадінні при рахунку вверх. Очистити OC1A/OC1B при співпадінні при рахунку вниз

Таблиця 13 – Режими роботи генератори форми сигналів

Режим	WGM13	WGM12 (CTC)	WGM11 (PWM11)	WGM10 (PWM10)	Режим таймера	TOP	Корекція OCR1x при	Прапор TOV1 встановити при
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x03FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fas t PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase	ICR1	TOP	BOTTOM

					Correct			
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

3.4. Конфігураційний регістр TCCR1B

Таблиця 14 – Формат регістра TCCR1B

Розряд	7	6	5	4	3	2	1	0
Ім'я	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Біт 7 – ICNC1: Фільтр шумів захоплення входу
Біт 6 – ICES1: Вибір фронту захоплення входу
Біт 5: Резерв
Біт 4:3 – WGM13:2: Режим генератора форми сигналу
Біт 2:0 – CS12:0: Джерело тактового сигналу

Вибір джерела тактового сигналу, а також запуск і зупинка таймера/лічильника здійснюється за допомогою розрядів CS12...CS10 регістра керування таймером TCCR1B. Відповідність між станом цих розрядів і режимом роботи таймера/лічильника показана в табл. 5.

Таблиця 15 – Джерело тактового сигналу для таймера/лічильника T1

CS12	CS11	CS10	Джерело тактового сигналу
0	0	0	Таймер/лічильник зупинено
0	0	1	Тактовий сигнал CLK (без дільника)
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	Виведення T1, інкремент лічильника по *спадаючому* фронту імпульсу
1	1	1	Виведення T1, інкремент лічильника по *наростаючому* фронту імпульсу

Регістри TCNT1, OCR1A/B, ICR1 є 16-розрядними, а шина даних 8-розрядна. Тому для доступу до них потрібні дві команди читання або запису:

```
...
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
```

```
in r16,TCNT1L
in r17,TCNT1H
...
```

3.5. Блок захоплення вхідних подій

Таймер лічильник T1 має блок захоплення вхідних подій і реєстрації часових позначок їх появи. Вхідний регістр ICR може захоплювати значення таймера/лічильника або вихід аналогового компаратора. Блок захоплення має цифровий фільтр від завад на вході. Зовнішні сигнали, які вказують на події, можуть подаватися на вивід ICP або через вихід блоку аналогового компаратора (ACO). Блок схема блоку захоплення показана на рис. 4.

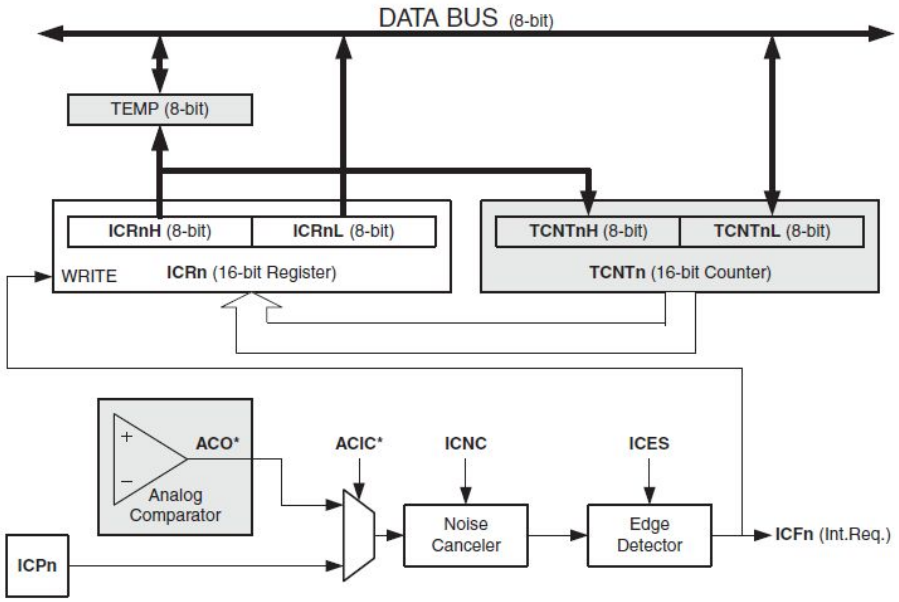


Рис. 4 – Схема блоку захоплення зовнішніх подій

4. Практична частина

4.1. Таймер T0 в режимі лічильника

При роботі таймера/лічильника T0 в режимі лічильника подій, подією буде замкнення кнопкою SW0 на платі STK500. Результат роботи програми відображається світлодіодами. 10-провідний шлейф на платі STK500 підключає виводи порту PB до виводів кнопок SW0-SW7 (в даному випадку буде використаний тільки вивід PB0, як вхід зовнішнього сигналу таймера T0). Інший 10-провідний шлейф з'єднує виводи порту PD з виводами світлодіодів LED0-LED7. Вивід порту PB0 програмується на ввід, а вивід порту PD0 – на вивід. Налаштувати таймер на режим рахування зовнішніх подій (натискання кнопки SW0). Після чотирьох, натискань переповнюється таймер (так як початкове значення лічильника TCNT0 – $FC) і викликається обробник переривань. Обробник включає світлодіод, показуючи, що програма виконана коректно, і знову ініціалізує лічильник таймера для продовження роботи. Тривалість включення світлодіода задається підпрограмою затримки.

Приклад 1.

```
;***********************************************
;1.asm - демонстрація роботи таймера T0 в режимі лічильника подій
;подія - натискання кнопки SW0 (PB.0).
;Світлодіоди включаються після четвертого натискання на кнопку SW0
;З'єднання шлейфами на платі STK500: порт PD-SW, порт PB-LED
;***********************************************
.include "m8515def.inc" ;файл визначень ATmega8515
.def temp = r16 ;тимчасовий регістр
.def Roz0 = r19
.def Roz1 = r20
.def Roz2 = r21
;** Таблиця векторів переривань
.org $000
rjmp INIT       ; обробка скидання
.org $007
rjmp T0_OVF     ; обробка переповнення таймера
;*** Ініціалізація МК
INIT: ldi temp,low(RAMEND) ; встановлення
out SPL,temp    ; вказівника стеку
ldi temp,high(RAMEND) ; на останню
out SPH,temp    ; комірку ОЗП
clr temp        ; ініціалізація порту
out DDRB, temp  ;  на вхід
ldi temp,0x01
out PORTB,temp  ; вкл. підтягуючого резистора PB.0/T0
ldi temp,0x01   ; ініціалізація порту PB.0
out DDRD,temp   ; на вихід
out PORTD,temp  ; виключення світлодіодів одиницями
ldi temp,0x20   ; SE=1 - дозвіл переходу
out MCUCR, temp ; в режимі Idle
;*** Налаштування таймера T0 на режим лічильника подій
ldi temp,0xff   ; чистка прапорів переривань
out GIFR,temp
out TIFR,temp
ldi temp,(1<<TOIE0) ; дозвіл переривань по переповненню таймера
out TIMSK, temp ;
ldi temp,0b0000_0111 ; рахувати по наростаючому фронту
out TCCR0,temp
sei             ; дозвіл переривань
ldi temp,0xfc   ; $FC=-04 для
out TCNT0,temp  ; рахунок чотирьох натискань
LOOP: sleep     ; перехід в режим пониженого
nop             ; енергоспоживання
rjmp LOOP
;*** обробка переривання при переповненні таймера T0
T0_OVF: clr temp
out PORTD,temp ; включення світлодіодів нулями
rcall DELAY    ; затримка
ldi temp,0x01
out PORTD,temp ; виключення світлодіодів
ldi temp,0xfc  ; $FC=-04 для
out TCNT0,temp ; рахунок чотирьох натискань
reti
;*** Затримка ***
```

```
DELAY:  ;  затримка розраховується
   ldi Roz1, 0x90
   ldi Roz0, 0xff
d1:
   subi Roz0,1
   sbci Roz1,0
   brcc d1
   ret
```

Події для таймера лічильника можна генерувати програмно. Для цього потрібно налаштувати вивід PB0 як вихід. В даному випадку інкремент лічильника буде відбуватися після виконання команд програми, які емулюють позитивний і негативний фронт, як цього вимагає налаштування таймера:

```
; позитивний фронт сигналу на PB0
cbi PORTB, 0
sbi PORTB, 0
; негативний фронт сигналу на PB0
sbi PORTB, 0
cbi PORTB, 0
```

Програма виконується на платі STK500 або в середовищі Proteus, рис. 5.

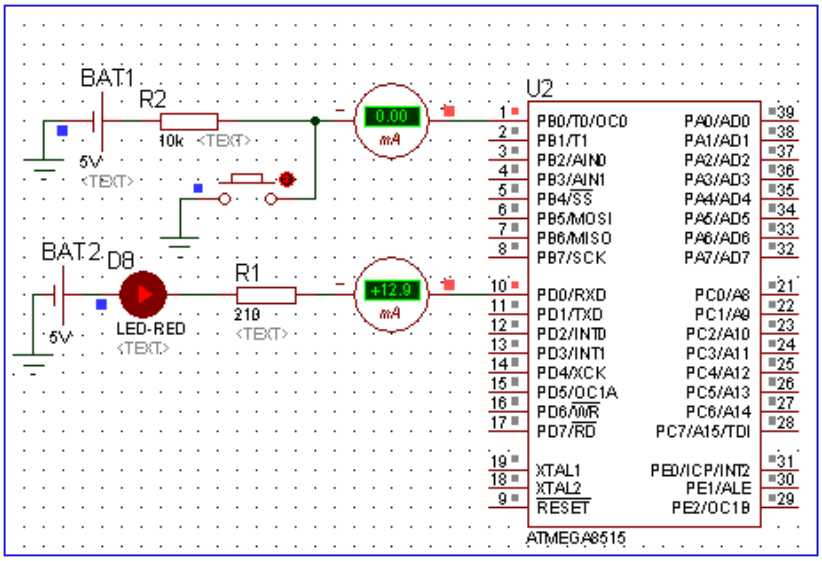


Рисунок 5 – Електрична схема з'єднання МК для дослідження роботи лічильника Т0

4.2. Таймер Т0 в режимі таймера

Для дослідження таймера/лічильника Т0 в режимі таймера виконуються наступні дії: при натисканні на першу кнопку на вхід таймера поступають сигнали з частотою, яка дорівнює частоті тактового генератора СК, при натисканні на другу кнопку на вхід таймера поступають сигнали з частотою, наприклад, СК/8. В обох випадках зразу після натискання засвічуються світлодіоди, а після переповнення таймера і обробки відповідного переривання світлодіоди гаснуть. Таким чином, у другому випадку час світіння світлодіодів буде у вісім разів більшим, ніж у першому, що і означає правильність виконання програми. Якщо встановити частоту тактового генератора МК рівною 512 Гц, то час включення світлодіодів в першому випадку 0,5 с, а у другому – 4 с (0,5-8). (Частоту тактового генератора потрібно змінювати після програмування МК!).

На платі STK500 потрібно з’єднати виводи порту PD з кнопками SW0 - SW7, виводи порту PB - із світлодіодами LED0 - LED7. Обробка натискання кнопок здійснюється послідовним опитуванням їх стану. Програма виконується на платі STK500 або в середовищі Proteus, рис. 6.

Приклад 2.

```
;*********************************************************
; 2.asm - демонстрація роботи таймера T0 в режимі таймера.
; Встановити частоту тактового генератора СК=512 Гц.
; При натисканні на SW2(PD2) на вхід поступають сигнали з частотою СК/64,
; При натисканні на SW3(PD3)- з СК/128.
; У першому випадку час до переповнення (виключення світлодіодів) - 0,5 с,
; у другому - 4 с.
; З'єднання: пара PD2:PD3-пара SW2:SW3, PB-LED
;*********************************************************
.include "m8515def.inc" /файл визначень ATmega8515
.def temp = r16 ; тимчасовий регістр
.equ SW2 = 2 ; 2-й вивід порту PD2/int0
.equ SW3 = 3 ; 3-й вивід порту PD3/int1
;***Таблиця векторів переривань
.org $000
rjmp INIT ; обробка скидання
.org $007
rjmp T0_Perep ; обробка переповнення таймера T0
; Ініціалізація МК
INIT: ldi temp,low(RAMEND) ; встановлення
out SPL,temp               ; вказівника стека
ldi temp,high(RAMEND)      ; на останню
out SPH,temp               ; комірку ОЗП
clr temp                   ;ініціалізація порту PD
out DDRD,temp              ; на ввід
ldi temp,0b0000_1100       ; включення підтягуючих
out PORTD,temp             ; резисторів порту PD
ser temp                   ; ініціалізація порту PB
out DDRB,temp              ; на виведення
out PORTB,temp             ; виключення світлодіодів
;***Налаштування таймера T0 на режим таймера
ldi temp,(1<<TOIE0)        ; дозвіл переривання по
out TIMSK,temp         ; переповненню таймера T0
clr temp
out TCCR0,temp         ; зупинити таймер таймер T0
sei
clr temp
out TCNT0,temp
;*** Очікування натискання кнопок
WAIT_SW2:
sbic PIND,SW2     ; перевірка натискання
rjmp WAIT_SW3     ; кнопки SW2 (skip if bit I/O cleared)
;*** Обробка натискання кнопки SW2
ldi temp,0x01     ; частота тактових сигналів- СК
rcall LED_ON
;rjmp WAIT_SW2
WAIT_SW3:
sbic PIND,SW3     ; перевірка натискання
rjmp WAIT_SW2
;*** Обробка натискання кнопки SW3
ldi temp,0x02     ; частота сигналів - СК/8
```

```
rcall LED_ON
rjmp WAIT_SW2

; **** Обробка переривання при переповненні таймера T0
T0_Perep:
ser temp
out PORTB,temp
clr temp
out TCCR0,temp ; зупинити таймер T0
clr temp
out TCNT0,temp ; запустити таймер з 0
;out TIFR,temp
reti

; *** Перемикання світлодіодів
LED_ON:
out TCCR0,temp        ; налаштування подільника тактового сигналу
clr temp              ;
out PORTB,temp
ret
```

Робота таймера/лічильника T1 в режимах лічильника подій і таймера аналогічна роботі таймера/лічильника T0. Відмінність полягає у розрядності базового лічильника: TCNT0 – 8-розрядний; TCNT1 – 16-розрядний. Тому програми для дослідження T1 в цих режимах можуть бути аналогічні раніше розглянутим.

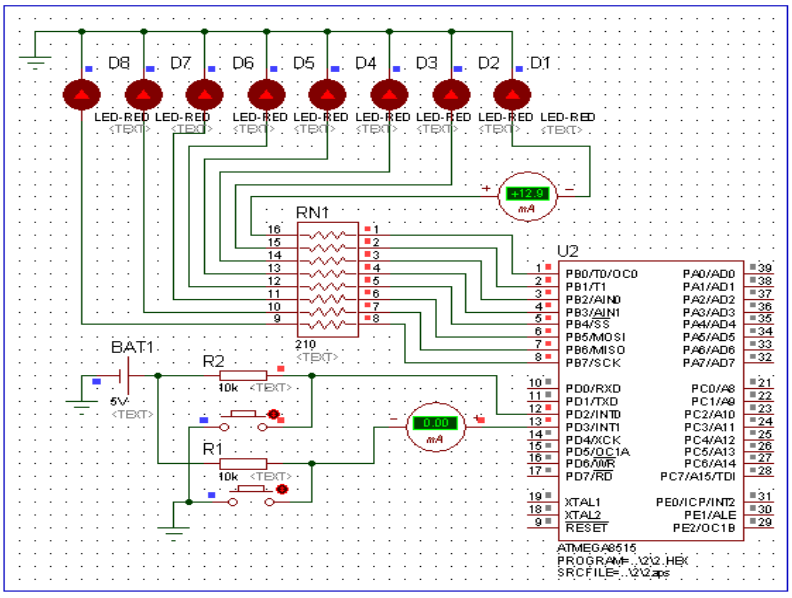


Рисунок 6 – Електрична схема з’єднання МК для дослідження роботи таймера T0

4.3. Таймер T1

4.3.1. Таймер T1 з функцією порівняння

Для реалізації у програмі функції порівняння використовується таймер/лічильник T1. При натисканні на кнопку START (SW0, PD0) запускається таймер – відбувається інкремент

лічильника з частотою CLK. При співпадінні значень лічильника TCNT1 і регістра порівняння OCR1B змінюється стан вивода OC1B на протилежний. При співпадінні значень лічильника TCNT1 і регістра порівняння OCR1A змінюється стану виводу OC1A на протилежний і скидається лічильний регістр в нульовий стан. Часові діаграми роботи таймера/лічильника T1 показані на рис. 7.

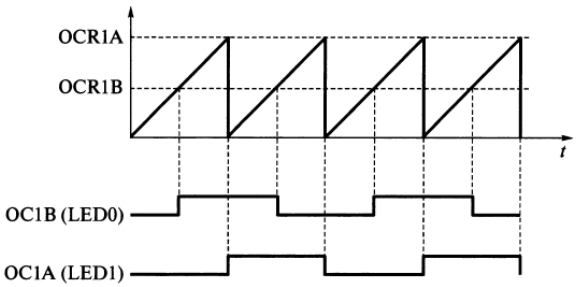


Рисунок 7 – Часові діаграми роботи таймера/лічильника T1 при роботі функцій порівняння

Можливість зупинки таймера під час рахунку реалізується за допомогою зовнішнього переривання від кнопки STOP (SWT, PD2). При записуванні значень в регістри порівняння спочатку записується старший байт, потім – молодший. Для спостереження змін стану виводів OC1A і OC1B їх необхідно з'єднати з виводами світлодіодів. Програма виконується в середовищі Proteus, рис. 8. Частота тактового генератора встановлюється рівною 256 Гц.

Приклад 3.

```
;**************************************
;3.asm - демонстрація роботи функції порівняння таймера T1.
;тактова частота генератора СК=256 Гц. При
;натискання SW0/(START, PD0) - інкремент лічильника з частотою СК
;натисканні SW2 (STOP,  PD2) - лічильник зупиняється.
; При співпадінні вмісту лічильника TCNT1 і
; регістра порівняння 0CR1A перемикається LED D0,
;              регістр порівняння  OCR1B - LED D1.
;З'єднання: LED0-PE2, LED1-PD5, SWO-PDO, SW1-PD2
;**********************************
.include "m8515def.inc" ;файл визначень ATmega8515
.def temp = r16         ;тимчасовий регістр
.def Roz0 = r19
.def Roz1 = r20
.def Roz2 = r21
.equ START = 0 ; PD0
.org $000
rjmp INIT ; $000 - обробка скидання
.org $001
rjmp EXT_INT0 ; $001 - зовнішнє переривання INT0 (натискання STOP)
.org $004
reti ;rjmp TIM1_COMPA ; $004 - Timer1 Компаратор A
.org $005
reti ;rjmp TIM1_COMPB ; $005 - Timer1 Компаратор B
.org $006
reti; rjmp TIM1_OVF
; ** ініціалізація МК
INIT:
ldi temp,low(RAMEND)       ; встановлення
```

```
out SPL,temp                ; вказівника стеку
ldi temp,high(RAMEND)       ; на останню
out SPH,temp                ; комірку ОЗП
ldi temp,0b0010_0000  ; 0,2 - на ввід, PD5/OC1A - вивід
out DDRD,temp   ;
ldi temp,0b0000_0101
out PORTD,temp
;  ініціалізація вивода порту
ldi temp,0x04   ; для ATmega8515
out DDRE, temp  ; PE2 (OC1B) на вивід
; *** Зовнішнє переривання ***
ldi temp,(1<<INT0) ; дозвіл переривання INT0
out GICR,temp
clr temp        ; від кнопки STOP (PD2) - INT0
out MCUCR, temp ; по низькому рівню (ISC01=0,ISC00=0)
;*** Налаштування функції порівняння таймера T1
ldi temp,0x00   ;
out TIMSK,temp  ; заборона переривань від таймера
cli             ; заборона переривань I
ldi temp,(1<<COM1A0|1<<COM1B0) ; TCNT1 порівнюється з  OCR1A, OCR1B
out TCCR1A, temp
clr temp
out TCCR1B, temp      ; таймер зупинено
ldi temp,0x00    ; запис числа в регістр порівняння OCR1A
out OCR1AH,temp  ; першим записується старший байт
ldi temp,0x80    ;
out OCR1AL,temp  ;
ldi temp,0x00    ; запис числа в регістр порівняння OCR1B,
out OCR1BH,temp  ;  першим записується  старший байт
ldi temp, 0xff   ;
out OCR1BL,temp  ;
sei              ; дозвіл переривань
ldi temp,(1<<OCIE1A|1<<OCIE1B)
out TIMSK,temp  ; дозвіл переривань TCNT1 при співпадінні ком.A або ком.B
WAITSTART:
sbic PIND,START   ;очікування натискання
rjmp WAITSTART  ; кнопки START
ldi temp,(1<<CS10)
out TCCR1B,temp
LOOP: nop       ; під час циклу
rjmp LOOP       ; збільшується вміст лічильника TCNT1
;*** Оброблення переривань INT0 від кнопки STOP
EXT_INT0:
  clr temp         ; обнулення
  out TCNT1H,temp  ; вмісту лічильного регістра TCNT1
  out TCNT1L,temp  ;
  ldi temp,0       ; зупинка таймера
  out TCCR1B,temp
WAITSTART_2:       ; очікування
  sbic PIND,START  ; натискання кнопки START
  rjmp WAITSTART_2
  ldi temp,(1<<CS10)
  out TCCR1B,temp
  reti
```

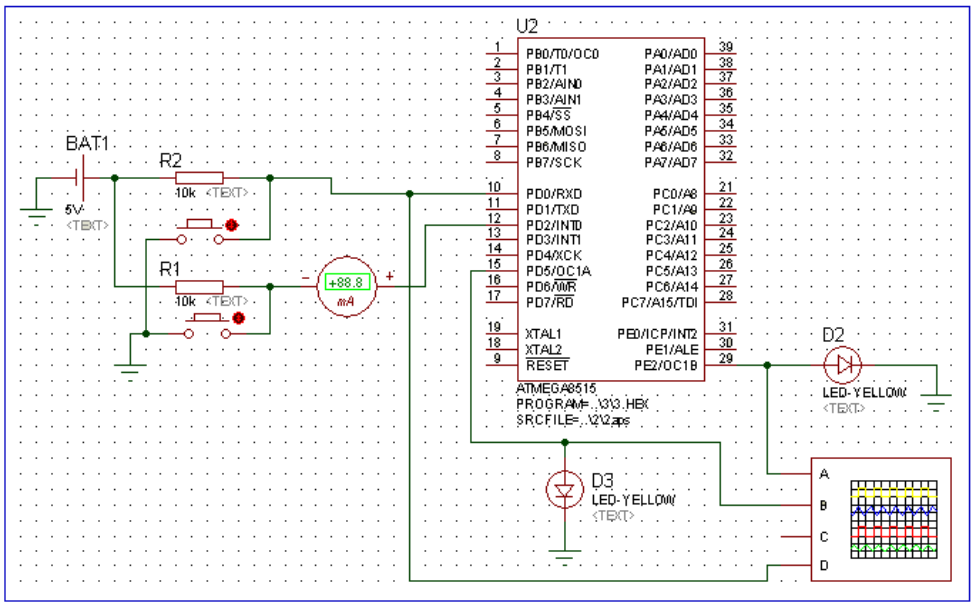


Рисунок 8 – Електрична схема з'єднання МК для роботи таймера Т1 в режимі порівняння

4.3.2. Таймер Т1 з функцією захоплення

Функція захоплення таймера/лічильника Т1 спрацьовує при натисканні кнопки СТАРТ. При захопленні викликається обробник переривань, який переписує вміст 16-розрядного регістра захоплення ICR1 в загальні регістри для зберігання молодшого і старшого байту (пересилання даних в порт для індикації можливе тільки з регістрів загального призначення). Вивід кожного байту зв'язаний з натисканням кнопок SW1, SW2. В регістрі порівняння OCR1A задається максимальне значення для лічильника і, отже, для регістра захоплення. Таймер налаштовується так, щоб він обнулявся при рівності значень лічильника TCNT1 і регістра порівняння OCR1A, встановленням біта WGM12 (CTC1) регістра TCCR1B в 1. Це використовується як додатковий признак правильності роботи програми. Так, наприклад, якщо вміст OCR1A дорівнює $00FF, а біт WGM12 (CTC1) встановлений в 1, то в старшому байті регістра захоплення завжди буде 0.

Вивід ICP в МК ATmega8515 – це лінія порту введення/виведення PE0. Виклик обробника переривань від події захоплення налаштовується встановленням прапора TICIE1 регістра маски TIMSK і прапора ICF1 регістра запитів TIFR.

Комутація кнопок з виводами МК здійснюється дводротовими провідниками: SW0 – PDO, SW1 – PD1, SW2 – PD2, SW3 – PE0. Світлодіоди підключають до виводів порту PB 10-дротовим шлейфом. Програма виконується на платі STK500 або в середовищі Proteus, рис. 9.

Приклад 4.

```
;*********************************************************
;4.asm (для МК ATMega8515) - функція захоплення таймера Т1.
;Частоту тактового генератора СК=256 Гц.
;Натискання на SW0 (START) - на вхід лічильника поступає сигнал СК
;Натискання на SW1 (SHOW_L) - на світлодіоди виводиться
;значення молодшого байту, а SW2 (SHOW_H) - старшого байта регістра ICR1.
;Натискання на SW3 (CAPT) - захоплення стану таймера.
;З'єднання: SW0-PD0, SW1-PD1, SW2-PD2, SW3-PE0,10-дротовим шлейфом PB-LED
;*********************************************************
```

```
.include "m8515def.inc"
.def temp = r16     ; тимчасовий регістр
.def H_byte = r17  ;для зберігання старшого байта
.def L_byte = r18  ;для зберігання молодшого байта
.equ START = 0     ;0-й вивід порту PD
.equ SHOW_L = 1    ;1-й вивід порту PD
.equ SHOW_H = 2    ;2-й вивід порту PD
;***Вектори переривань
.org $000
rjmp INIT          ; обробка скидання
.org $003
rjmp TIM1_CAPT ; обробка переривання по сигналу захоплення (SW3)
;***ініціалізація МК
INIT: ldi temp,low(RAMEND) ; встановлення
out SPL,temp               ; вказівника стека
ldi temp,high(RAMEND)      ; на останню
out SPH,temp     ; комірку ОЗП
ldi temp,0x00    ; ініціалізація
out DDRD,temp    ; порту PD на ввід
ldi temp,0x07    ; включення підтягуючих
out PORTD,temp   ; резисторів порту PD0,PD1,PD2
ldi temp,0x00
out DDRE,temp    ; ініціалізація PE0 (ICP) на ввід
ser temp
out DDRB,temp    ; ініціалізація порту PB на вивід
out PORTB,temp   ; виключення світлодіодів
cli
ldi temp, 0x00   ; відключення від таймера виводів
out TCCR1A,temp
ldi temp, 0x00    ;
out TCCR1B,temp  ; таймер T1 зупинений
ldi temp,0xff    ;
out OCR1AH,temp  ; запис числа в регістр порівняння OCR1A,
ldi temp,0xff    ; першим записується старший байт
out OCR1AL,temp
clr temp
out TCNT1H,temp  ; обнулення вмісту
out TCNT1L, temp ; лічильного регістру
ldi L_byte,0xF0  ; початкове значення (1111_0000)
ldi H_byte,0x0F  ; для контролю       (0000_1111)
sei              ; дозвіл переривань
WAITSTART:
sbic PIND,START      ;очікування натискання
rjmp WAITSTART   ; кнопки START (SW0)
ldi temp,(1<<TICIE1)         ; дозвіл переривання
out TIMSK,temp               ; по події захоплення таймера
ldi temp,(1<<WGM12|1<<CS10)  ; запуск лічильника, при
out TCCR1B,temp              ; співпадінні з OCR1A - скидування

WAIT_L: sbic PIND,SHOW_L     ;очікування натискання кнопки SW1
rjmp WAIT_H      ; для показу молодшого байту
out PORTB,L_byte ;виведення на світлодіоди

WAIT_H: sbic PIND,SHOW_H ; очікування натискання кнопки SW2
rjmp WAIT_L      ; для показу старшого байта
```

```
out PORTB,H_byte ;вивід на світлодіоди
rjmp WAIT_L
;*** Обробка переривань від кнопки CAPT (PE0)
TIM1_CAPT:
in L_byte,ICR1L  ;зчитування молодшого байту ICR1
in H_byte,ICR1H  ;зчитування старшого байту  ICR1
com L_byte       ;інвертування
com H_byte       ;інвертування
reti
```

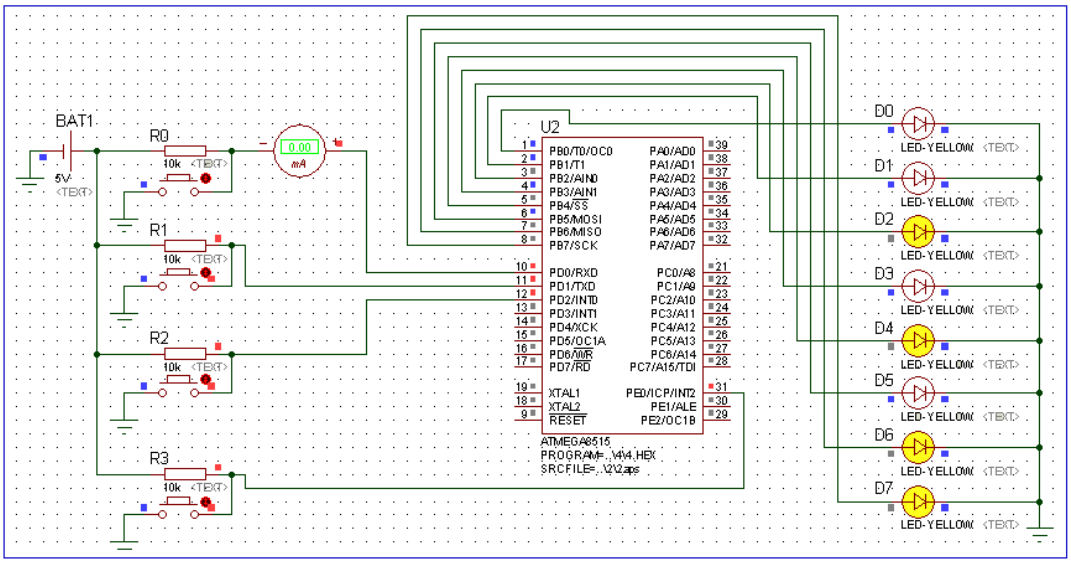


Рисунок 9 – Електрична схема з’єднання МК для роботи лічильника Т1 в режимі захоплення

4.3.3. Таймер Т1 у режимі ШІМа

При роботі лічильника Т1 в режимі широтно-імпульсного модулятора (ШІМа) використовуються виводи порту PD0-PD3, як кнопки SW0-SW3 відповідно. Варіанти обробки натискання кнопок:

1) при натисканні SW0 на виходах OC1A і OC1B встановлюється значення 0 и 1 відповідно (світиться LED0);

2) при натисканні SW1 відбувається генерація ШІМ-сигналу із шпаруватістю F1 (шпаруватість залежить від значення в регістрі порівняння) і світлодіоди позмінно включаються/виключаються;

3) при натисканні SW2 відбувається генерація ШІМ-сигналу із шпаруватістю F2 і світлодіоди позмінно включаються/виключаються (світиться LED1);

4) при натисканні SW3 на виходах OC1A і OC1B встановлюються 1 і 0 відповідно.

Часова діаграма роботи ШІМа показана на рис. 8.

Виводи OC1A (PD5) і OC1B (PE2) підключаються до світлодіодів (PD5-LED0, PE2-LED1). Програма виконується на платі STK500 або в середовищі Proteus, рис. 10.

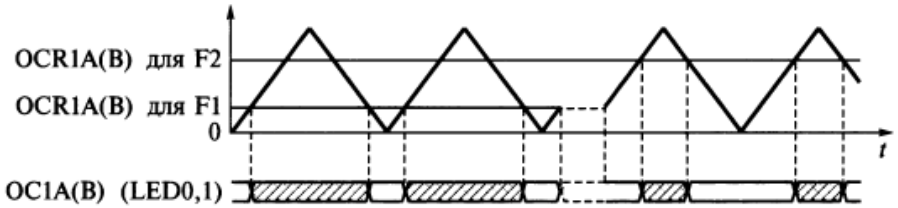


Рисунок 10 – Формування ШІМ-сигналу

Приклад 5.

```
;*******************************************************
; 5.asm - роботи таймера T1 в режимі ШІМа.
; Частота тактового генератора 2048 Гц.
; Натискання SW0 (SHOW_0) на виходах OC1A і OC1B встановлюється 0 і 1,
; SW1 (SHOW_F1) - генерація ШІМ-сигналу із шпаруватістю F1,
; SW2 (SHOW_F2) - із шпаруватістю F2,
; SW3 (SHOW_1)- на виходах встановлюється 1 і 0.
; З'єднання: PD5-LED0, PE2-LED1, PD0:PD1-SW0:SW1, PD2:PD3-SW2:SW3
;*************************************************************
.include "m8515def.inc"  ; файл визначень ATmega8515
.def temp = r16          ; тимчасовий регістр
; *** виводи порту PD
.equ SHOW_0 = 0
.equ SH0W_F1 = 1
.equ SHOW_F2 = 2
.equ SHOW_3 = 3
.org $000
rjmp INIT                       ; обробка скидання
;***Ініціалізація МК
INIT: ldi temp,low(RAMEND)      ; встановлення
out SPL,temp                    ; вказівника стеку
ldi temp,high(RAMEND)           ; на останню
out SPH,temp                    ; комірку ОЗП
ldi temp,0x20                   ; ініціалізація виводів порту PD:
out DDRD,temp                   ; 0-3 - на ввід, 5 - на вивід
ldi temp,0x0f                   ; включення підтягуючих
out PORTD, temp                 ; резисторів порту PD
ldi temp,0x04                   ; для ATmega8515 ініціалізація
out DDRE, temp                  ; PE2 (OC1B) на вивід
cli                             ; заборона переривань
;ldi temp,0xB3                   ; налаштування таймера на ШІМ
ldi temp,(1<<COM1A1|1<<COM1B1|1<<COM1B0|1<<WGM11|1<<WGM10)
out TCCR1A,temp                 ; з виводами OC1A і OC1B
clr temp
out OCR1AH,temp                 ; обнулити
out OCR1AL,temp                 ; регістри порівняння
out OCR1BH,temp
out OCR1BL,temp
out TCNT1H,temp                 ; регістра лічильного
out TCNT1L,temp
ldi temp,0x01
out TCCR1B,temp                 ; запустити таймер: частота - CK
sei                             ; дозвіл переривань
WAIT_0:                         ; очікування
```

```
sbic PIND, SHOW_0              ; натискання
rjmp WAIT_F1                   ; кнопки SHOW_0 (PD0), стан 0 і 1
;***Переведення в стійкий стан виводів OC1A=0, OC1B=1
clr temp                       ;запис числа в
out OCR1AH,temp                ; регістри порівняння, першим
out OCR1AL,temp                ; записується старший байт
out OCR1BH,temp
out OCR1BL,temp
WAIT_F1:sbic PIND,SH0W_F1  ; очікування натискання
rjmp WAIT_F2                   ; кнопки SH0W_F1 (PD1) - режим ШІМа
;***Налаштування таймера на режим ШІМа із шпаруватістю F1
ldi temp,0x00                  ; запис числа в
out OCR1AH,temp                ; регістри порівняння,
out OCR1BH,temp                ; першим записується
ldi temp, 0xFF                 ; старший байт
out OCR1AL,temp
out OCR1BL,temp
WAIT_F2:sbic PIND, SHOW_F2 ; очікування натискання
rjmp WAIT_3                    ; кнопки SHOW_F2 (PD2) - режим ШІМ
;***Налаштування таймера на режим ШІМ із шпаруватістю F2
ldi temp,0x02                  ; запис числа в
out OCR1AH,temp                ; регістри порівняння,
out OCR1BH,temp                 ; першим записується
ldi temp, 0xFF                  ; старший байт
out OCR1AL,temp
out OCR1BL,temp
WAIT_3:sbic PIND,SHOW_3      ; очікування натискання PIND 0 16
rjmp WAIT_0                    ; кнопки SHOW_3, стан 1 і 0
;***Переведення в стійкий стан виводів OC1A=1, OC1B=0
ser temp                       ; запис числа в
out OCR1AH,temp                ; регістри порівняння, першим
out OCR1AL,temp                ; записується старший байт
out OCR1BH,temp
out OCR1BL,temp
rjmp WAIT_0
```

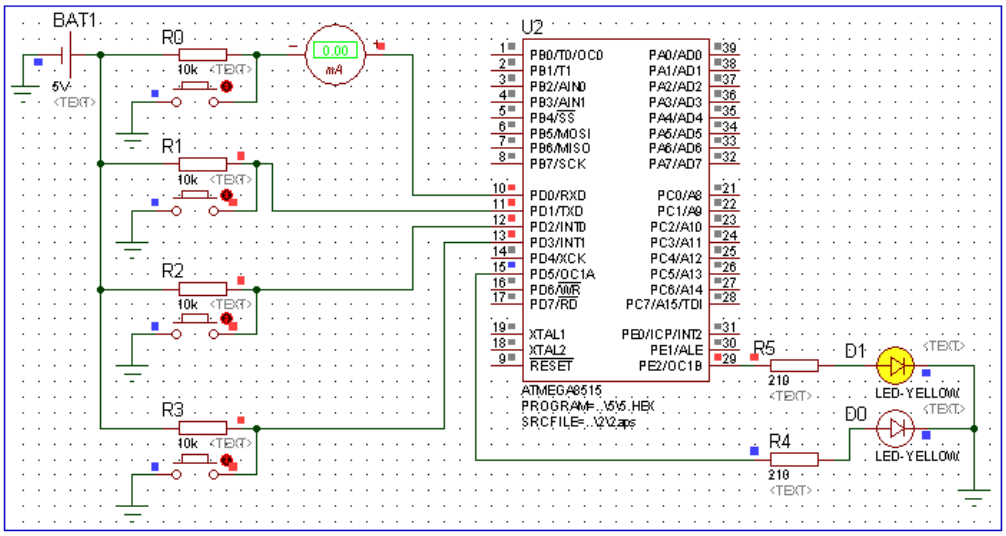


Рисунок 11 – Електрична схема з’єднання МК для роботи лічильника Т1 в режимі ШІМа

4.4. Сторожовий таймер

Основна функція сторожового таймера (Watchdog Timer) – перервати виконання зацикленої програми або вийти з інших непередбачуваних ситуацій. Сторожовий таймер тактується від окремого вбудованого на кристалі тактового генератора частотою 1 МГц. За допомогою подільник сторожового таймера інтервал часу до перевантаження МК. Схема сторожового таймера показана на рис. 12.

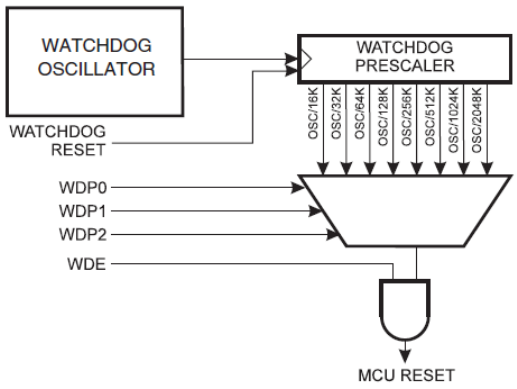


Рисунок 12 – Схема сторожового таймера

Для керування сторожовим таймером призначений регістр WDTCR. Формат регістра показаний в табл. 16.

Таблиця 16 – Формат регістра WDTCR

Розряд	7	6	5	4	3	2	1	0
Ім’я	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0

WDCE – дозвіл на виключення сторожового таймера.

WDE – дозвіл (1)/заборона(0) використання сторожового таймера.

WDP2-WDP0 – масштабування тактового генератора.

Для включення/виключення сторожового таймера використовують два розряди регістра WDE і WDCE. Якщо розряд WDE встановлений в 1, таймер дозволено використовувати, якщо скинутий в 0 – заборонено.

Для запобігання випадкового виключення таймера призначений розряд WDCE. Справа в тому, що виключення сторожового таймера (скидання розряду WDE) можна здійснити тільки при встановленому розряді WDCE, який через чотири машинних такти після встановлення в 1 апаратно скидається. Завдяки цьому можливість випадкового виключення сторожового таймера практично виключається.

Виходячи із цього, для виключення сторожового таймера рекомендується наступний порядок дій:

– однією командою записати 1 в розряди WDE і WDCE;

– за наступні чотири машинні тактів записати 0 в розряд WDE.

Період наступлення тайм-аута сторожового таймера задається за допомогою розрядів WDP2-WDP0 регістра WDTCR згідно табл. 17.

Щоб запобігти випадковому скиданню МК при зміні періоду сторожового таймера, необхідно перед записом розрядів WDP2-WDP0 заборонити роботу таймера або скинути його командою RESET.

Таблиця 17 – Час перевантаження МК (Reset) від сторожового таймера

WDP2	WDP1	WDP0	Число тактів сторожового таймера	Період до Reset	
				V_{CC}=3,0 В	V_{CC}=5,0 В
0	0	0	16·K (16,384)	17,1 мс	16,3 мс
0	0	1	32·K (32,768)	34,3 мс	32,5 мс
0	1	0	64·K (65,536)	68,5 мс	65 мс
0	1	1	128·K (131,072)	0,14 с	0,13 с
1	0	0	256·K (262,144)	0,27 с	0,26 с
1	0	1	512·K (524,288)	0,55 с	0,52 с
1	1	0	1024·K (1,048,576)	1,1 с	1,0 с
1	1	1	2048·K (2,097,152)	2,2 с	2,1 с

Приклад налаштування сторожового таймера. Вважається, що переривання контролюються (наприклад, глобальною забороною), тому переривання трапиться при виконанні цієї функції.

```
WDT_off:
; Write logical one to WDCE and WDE
ldi r16, (1<<WDCE)|(1<<WDE)
out WDTCR, r16
; Turn off WDT
ldi r16, (0<<WDE)
out WDTCR, r16
ret
```

4.4.1. Програмування сторожового таймера

Вважаючи, що при натисканні на кнопку SW0 тайм-аут наступить через 0,49 с, а при натисканні на SW1 – через 1,9 с, вибрати з табл. 17 (V_{CC} = 5 В) значення завантажуваних коефіцієнтів згідно заданого періоду сторожового таймера. В обох випадках до наступлення переверви світлодіоди повинні бути включені, після чого скидається МК, і світлодіоди гаснуть. Програма для МК приведена нижче. (Увага! Перед програмуванням МК ATmega8515 у вікні STK500 AVR Studio 4 на закладці Fuses встановити прапор S8515C і запрограмувати конфігураційну комірку.)
Комутація виводів: SW0 – PDO, SW1 – PD1, LED – PB 10-дротовим шлейфом.

Приклад 6.

```
;*****************************************************
; 6.asm - робота сторожового таймера із незалежним генератором.
;Натискання SWO (PERIOD_l) тайм-аут наступає через 0,49 с (включення
; світлодіодів).
; Натискання SW1 (PERIOD_2) - через 1,9 с.
;З'єднання: PDO:PD1-SW0:SW1, PB-LED(10-дротовий шлейф)
;.*****************************************************
```

```
.include "m8515def.inc" ;файл визначень ATmega8515
.def temp = r16         ;тимчасовий регістр
;***Виводи порту PD
.equ PERIOD_1 = 0
.equ PERIOD_2 = 1
.org $000
rjmp INIT                    ; обробка скидання
;* * *Ініціалізація МК
INIT: ldi temp,low(RAMEND)   ; встановлення
out SPL,temp                 ; вказівника стека
ldi temp,high(RAMEND)        ; на останню
out SPH,temp                 ; комірку ОЗП
clr temp                     ; ініціалізація порту PD
out DDRD,temp                ; на введення
ldi temp,0x03                ; включення підтягуючих
out PORTD,temp               ; резисторів порту PD0,PD1
ser temp                     ; ініціалізація порту PB
out DDRB,temp                ; на виведення
out PORTB,temp               ; виключення світлодіодів PB
ldi temp,(1<<WDCE|1<<WDE)    ; виключення
out WDTCR, temp              ; сторожового таймера:
ldi temp,(1<<WDCE)           ; WDE=0
out WDTCR,temp
WAIT_SW0: sbic PIND,PERIOD_1 ; очікування натискання
rjmp WAIT_SW1                ; кнопки PERIOD_1
;*** тай-аут 0,49 с
clr temp                     ; включення/виключення світлодіодів
out PORTB,temp               ;
ldi temp,$0D                 ; включення сторожового таймера,
out WDTCR,temp               ; період 0,49 с
WAIT_SW1: sbic PIND,PERIOD_2 ;очікування натискання
rjmp WAIT_SW0                ; кнопки PERIOD_2
;*** тайм-аут 1,9 с
clr temp                     ; включення/виключення світлодіодів
out PORTB,temp               ;
ldi temp,$0F                 ; включення сторожового таймера,
out WDTCR,temp               ; період 1,9 с
rjmp WAIT_SW0
```

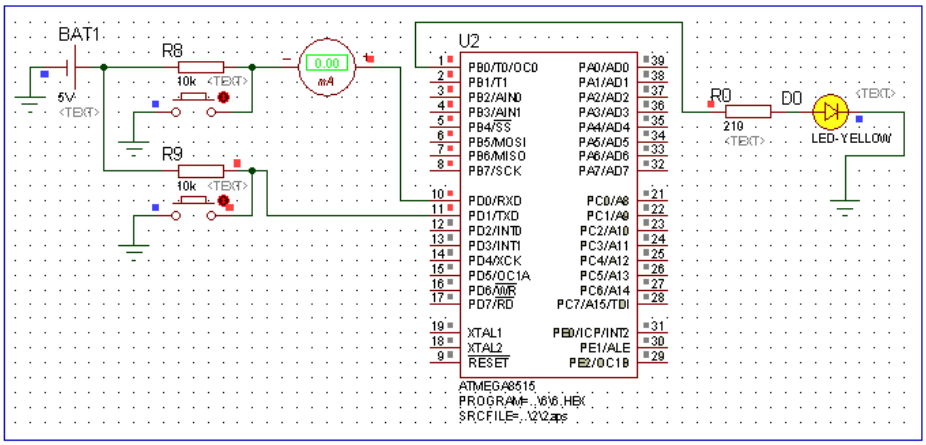


Рисунок 13 – Електрична схема з’єднання МК для роботи зі сторожовим таймером

Завдання.

1. Написати програму (на основі прикладу 1) в якій таймер/лічильник Т0 працює в режимі лічильника подій. Лічильник подій має підраховувати натискання кнопки і значення виводити на лінійку з 8-ми світлодіодів.

2. Написати програму (на основі прикладу 2) в якій таймер/лічильник Т0 працює в режимі лічильника подій. Лічильник подій має підраховувати натискання кнопки і значення кратне 10 виводити на лінійку з 4-х світло діодів.

3. Змінити програму прикладу 1 так, щоб при натисканні кнопки (SW1, PD0) виводилося на індикатор число зареєстрованих подій.

4. Змінити програму прикладу 2 для перевірки роботи таймера/лічильника Т1 в режимі таймера. Задати час включення світлодіодів 0.1 і 5 сек.

5. Змінити програму прикладу 2 для перевірки роботи таймера/лічильника Т1 в режимі таймера. Задати час включення світлодіодів 1 і 10 сек.

6. Написати програму, яка кожні 50 мс по запиту переривання від таймера зчитує стан кнопкового регістра SW, визначає номер натиснутої кнопки і висвічує його світлодіодами.

7. Написати програму електронного секундоміра, передбачивши виведення секунд в молодшу тетраду, десятків секунд – в старшу тетраду світлодіодної лінійки. Для відрахунку часу використовувати 16-розрядний таймер/лічильник Т1, при переповненні якого формується переривання і викликається обробник переривань, який накопичує таймерні інтервали часу і оновлює показання світлодіодного індикатора.

8. Написати програму (на основі прикладу 3) в якій однією кнопкою зупиняється, а іншою кнопкою запускається таймер лічильник Т1. При роботі лічильника засвічувати світлодіод.

9. Написати програму (на основі прикладу 4) в якій використовується функція захоплення таймера Т1. Функція захоплення таймера/лічильника Т1 спрацьовує при натисканні кнопки SW0. При захопленні (кнопка SW3) викликається обробник переривань, який переписує вміст 16-розрядного регістра захоплення ICR1 в загальні регістри для зберігання молодшого і старшого байту. Вивід кожного байту зв’язаний з натисканням кнопок SW1, SW2. В регістрі порівняння OCR1A задається максимальне значення 64 для лічильника і, отже, для регістра захоплення.

10. Написати програму (на основі прикладу 5) в якій використовується функція ШІМ таймера Т1. Запуск і зупинка таймера кнопками SW0, SW1. Значення шпаруватості сигналу задавати кнопкою SW2. Для індикації ШІМ сигналу використати світлодіод D1.

11. Написати програму сторожового таймера із незалежним генератором, який спрацьовує через 0.1, 0.5, 2 сек. Час спрацювання сторожового таймера задавати кнопками SW1, SW2, SW3.

Примітка. Номер варіанта завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Текст програми із поясненнями.
4. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми.

Питання.

1. Які виводи використовують таймери/лічильники МК ATmega8515?
2. Розказати про структурну схему таймера/лічильника Т0?

3. Які функції може виконувати таймер/лічильник T0?

4. Розказати про структурну схему таймера/лічильника T1?

5. Які функції може виконувати таймер/лічильник T1?

6. Пояснити роботу таймера/лічильника T1 в режимі таймера?

7. Пояснити роботу таймера/лічильника T1 в режимі захоплення?

8. Пояснити роботу таймера/лічильника T1 в режимі порівняння?

9. Пояснити роботу таймера/лічильника T1 в режимі ШІМ?

10. Який максимальний час захоплення можна зареєструвати при роботі програми прикладу 4?

11. Що таке шпаруватість сигналів? Як змінюється часова діаграма вихідного сигналу при зміні значення, яке завантажується в регістр порівняння в режимі ШІМ?

12. Які значення необхідно занести в регістри порівняння в програмі прикладу 5 для формування ШІМ-сигналів шпаруватістю 4 (відношення періоду до тривалості сигналу) при тому ж періоді ШІМ-сигналів?

13. Який максимальний час наступання перерви у сторожового таймера ATmega8515?

ЛАБОРАТОРНА РОБОТА № 9. Послідовний обмін даними по каналу USART

Мета роботи – вивчення універсального синхронного/асинхронного приймача/передавача USART та програмування введення/виведення.

1. Інтерфейс USART

МК ATmega мають вбудований повнодуплексний універсальний синхронно/асинхронний приймач-передавач USART. Через нього можна приймати і передавати інформацію задану послідовним кодом.

При *синхронному* послідовному введенні/виведенні передача окремих бітів даних синхронізується за допомогою тактового сигналу, який передається одночасно з даними. Синхронна послідовна передача даних використовується, основним чином, на рівні друкованих плат, у тому числі, для обміну даними між різними інтегрованими блоками у складі схеми МК та різними периферійними схемами.

При *асинхронній* передачі даних синхронізація виконується у часі за допомогою стартових та стопових бітів, що визначають початок та кінець передачі слова даних. Асинхронна передача даних використовується для комунікації блоків, розділених у просторі та які мають певну автономність один від одного. Наприклад, між ПК та принтером, між пристроєм на базі МК та комп'ютером.

Частіше використовується асинхронний режим роботи, тому далі мова йтиме про асинхронний режим передачі даних (UART). Інтерфейс UART дводротовий, лінія TxD призначена для передачі даних, а RxD – для прийому даних. В проміжку між пересиланням даних лінія TxD знаходиться в стані лог. 1, а на лінії RxD має бути лог.0, встановлений зовнішнім передавачем. Звичайно вважається, що лінії RxD і TxD під'єднані до перетворювача логічних рівнів UART в сигнали відповідного інтерфейсу (RS-232 або RS-485). Тому на відміну від SPI і I^2C, UART використовується для зв'язку віддалених пристроїв (до декількох сотень метрів) між собою.

Структурна схема USART показана на рис. 1.

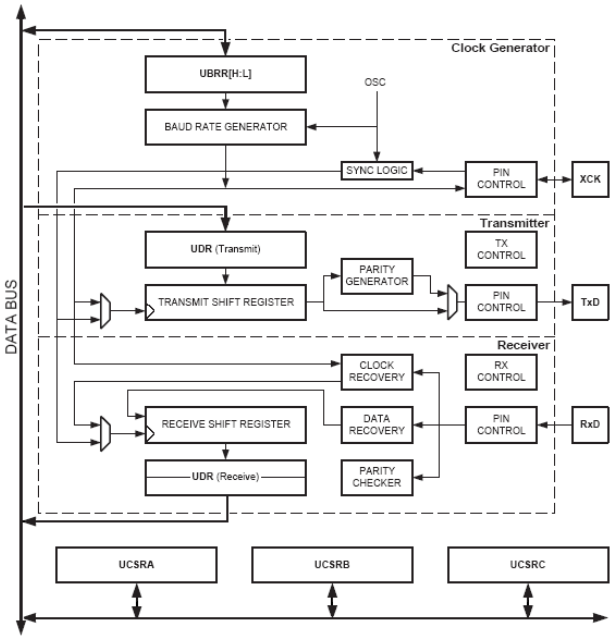


Рисунок 1 – Структурна схема USART

Дані в протоколі UART передаються кадрами.

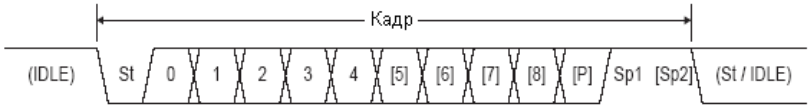


Рисунок 2 – Формат кадру

Кожний кадр містить один стартовий біт (St, завжди лог.0), біти даних (від 5 до 9), біт контролю парності або непарності (P, необов'язковий), один або два стопових біти (SP1, SP2, завжди лог.1). При відсутності передачі (режим очікування, IDLE) лінія має стан лог. 1. Стартовий і стоповий біти використовуються для синхронізації передачі даних. Наступна передача даних може відбутися через довільний проміжок часу, тому протокол і називається асинхронним.

Відомо, що при передачі даних можуть відбутися різні збої. Модуль USART при прийманні може виявити помилки переповнення даних і формату кадра.

Для взаємодії з програмою в модулі передбачені переривання при виникненні наступних подій: приймання закінчено (RX complete) з адресою вектора $009 в таблиці векторів переривань, регістр даних передавача UDR0 порожній (TX data register empty) з адресою вектора $00a, передавання закінчено (TX complete) з адресою вектора $00b.

2. Регістри UART

USART використовує як вхід приймача RXD вивід PD0, а як вихід передавача TXD – вивід PD1. Дані, які приймаються і передаються зберігаються в регістрі UDR, рис 3. Фізично регістр UDR складається з двох окремих регістрів, один із яких RXB (Receive data buffer register) використовується для приймання даних, а інший TXB (Transmit data buffer register), – для передачі. При читанні (Read) регістра UDR звертаються до регістра приймача, при записуванні (Write) – до регістра передавача,. Якщо передається 5, 6 або 7 бітів то решта бітів ігнорується передавачем і встановлюється в нуль приймачем.

Біт	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рисунок 3 – Регістр даних UDR

Керують роботою і відображають статус USART регістри UCSRA, UCSRB, UCSRC, рис. 4 – 6. Швидкість передачі задається регістром UBRR (UBRRH, UBRRL), старший байт якого має ту саму адресу, як і регістр UCSRC, рис. 7

Біт	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Початкове значення	0	0	1	0	0	0	0	0	

Рисунок 4 – Формат регістра UCSRA: RXC – прийом закінчений (лог.0), TXC – передача закінчена (лог.1), UDRE – регістр даних порожній (лог.1), FE – помилка формату кадра (лог.1), DOR – переповнення даних (лог.1), PE – помилка парності/непарності бітів (лог.1), U2X – подвоєння швидкості передачі в асинхронному режимі (лог.1), синхронний режим (лог.0), MPCM – режим багатопроцесорних комунікацій (лог.1)

Біт	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рис. 5 – Формат регістра UCSRB: RXCIE – дозвіл переривань по прапору RXC (лог.1), TXCIE – дозвіл переривань по прапору TXC (лог.1), UDRIE – дозвіл переривань по прапору UDRE (лог.1), RXEN – приймач доступний (лог.1), TXEN – передавач доступний (лог.1), UCSZ2 – число бітів даних, RXB8 – отримання 8-біта, TXB8 – передача 8-біта

Біт	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	1	0	0	0	0	1	1	0	

Рис. 6 – Формат регістра UCSRC: URSEL – вибір регістра (UCSRC – лог.1, UBRRH – лог.0), UMSEL – вибір режиму (асинхронний – лог.0, синхронний – лог.1), UPM1,UPM0 – режим контролю парності/непарності (00 – недоступний, 10 – контроль парності, 11 – контроль непарності), USBS – вибір кількості стоп бітів (лог.0 – 1 біт, лог.1 – два біти), UCSZ2,UCSZ1,UCSZ0 – кількість бітів даних (000 – 5 бітів, 001 – 6 бітів, 010 – 7 бітів, 011 – 8 бітів, 111 – 9 бітів, UCPOL – полярність тактового сигналу для синхронного режиму

Біт	15	14	13	12	11	10	9	8	
	URSEL				UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рис. 7 – Формат регістра UBRR: URSEL – вибір регістра UBRRH (лог.0), UBRR[0:11] – задання швидкості обміну

USART підтримує чотири режими роботи тактових сигналів для передавача і приймача: нормальний асинхронний, асинхронний із подвоєною швидкістю, синхронний режим ведучого та синхронний режим веденого. Швидкість обміну (біт/с) в нормальному асинхронному режимі задається формулою

$$\text{BOUND} = \frac{f_{\text{osc}}}{16(\text{UBRR}+1)},$$

де f_{osc} – частота осцилятора, UBRR – значення бітів UBRR[0:11].

3. Ініціалізація UART

Перед роботою USART потрібно задати швидкість обміну, формат кадру і потрібний стан (приймач або передавач). Якщо робота USART буде керуватися перериваннями, то прапор SREG.I необхідно очистити. При заданні режиму обміну в регістрі UCSRC необхідно обов'язково встановити біт URSEL. Приклад ініціалізації асинхронної передачі без використання переривань і фіксованим форматом кадру:

```
USART_Init:
out UBRRH,r17   ; встановлення швидкості обміну з табл. 1
out UBRRL,r16
ldi r16, (1<<RXEN)|(1<<TXEN)  ; дозвіл стану приймача і передавача
out UCSRB,r16
Ldi r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0) ; формат кадру: 8data, 2stop bit
out UCSRC,r16
ret
```

4. Передача даних

Передавач USART стає доступним при встановленні розряду TXEN регістра UCSRB. У цьому випадку вивід PD1 підключається до передавача USART і починає функціонувати як вихід TxD незалежно від стану 1-го розряду регістра DDRD порта PD.

Передача ініціюється записом даних у буферний регістр даних UDR. Після цього дані пересилаються з регістра UDR у зсувний регістр передавача, якщо він у стані очікування або зразу ж після передачі останнього стоп-біта попереднього кадру. Після пересилання вмісту регістра UDR у зсувний регістр прапор UDRE регістра UCSRA встановлюється в 1, що означає готовність передавача до отримання нового значення. В цьому стані прапор залишається до нового запису в регістр UDR. Після завантаження нових даних у зсувний регістр передача здійснюється згідно встановленого режиму обміну. Одночасно з пересиланням формується службова інформація: 0-й розряд зсувного регістра скидається в 0 (старт-біт), а 9-й (або 10-й) розряд встановлюється в 1 (стоп-біт). Якщо задано режим передачі 9-розрядних даних, то значення розряду TXB8 регістра UCSRB копіюється в 9-й розряд зсувного регістра.

Після завантаження зсувного регістра його вміст зсовується вправо і поступає на вивід TXD в наступному порядку: стартовий біт, дані (починаючи з молодшого розряду), стоповий біт. Зсув здійснюється по тактовому сигналу, який виробляє контролер швидкості передачі. Якщо встановлений біт парності/непарності UPM1 регістра UCSRC то після останнього біта даних вставляється відповідний біт, обчислений блоком контролю передачі. Після закінчення передачі усього кадру і відсутності нових даних в буферному регістрі UDR встановлюється прапор TXC в регістрі UCSRA.

Підпрограма пересилання даних очікує очищення регістра UDR шляхом перевірки прапора UDRE. Якщо використовується переривання UDRIE, то підпрограма переривання записує дані у буферний регістр UDR:

```
USART_Transmit:
sbis UCSRA,UDRE  ;очікування очищення буферного регістра UDR
rjmp USART_Transmit
out UDR,r16      ;завантаження даних у регістр UDR, відсилання даних
ret
```

5. Прийом даних

Приймач USART стає доступним при встановленні розряду RXEN регістра UCSRB. Після виявлення старт-біта починається оброблення поступаючих розрядів слова даних. Рішення про значення прийнятого розряду приймається за результатом трьох вибірок вхідного сигналу в середині бітового періоду. Станом розряду вважається логічне значення, яке було отримано в двох з трьох вибірок. По мірі розпізнання розрядів прийманої послідовності вони поступають, зсовуючись вправо, у зсувний регістр приймача.

Стоп-біт також розпізнається по трьох вибірках вхідного сигнала. Стоп-біт вважається прийнятим, якщо значення хоча б двох з трьох вибірок вхідного сигналу дорівнюють 1. Інакше фіксується помилка кадру і встановлюється в 1 прапор FE регістра UCSRA. Перед читанням регістра даних UDR потрібно завжди перевіряти стан цього прапора.

Після отримання першого стоп-біта вміст зсувного регістра пересилається в буфер отримувача UDR. При прийомі 9-розрядного кадру, 9-біт необхідно прочитати з розряду RXB8 регістра UCSRB, до читання бітів з регістра UDR.

Підпрограма приймання даних опитує прапор RXC регістра UCSRA і якщо він встановлений отримує і повертає дані:

```
USART_Receive:
sbis UCSRA, RXC    ; очікування на отримання даних
rjmp USART_Receive
in r16, UDR        ; зчитування і повернення даних з буфера
ret
```

Якщо в буферний регістр зчитані дані то встановлюється прапор RXC регістра UCSRA. При встановленому прапорі переривання RXCIE регістра UCSRB викликається підпрограма оброблення переривання, яка зчитує дані з регістра UDR і тим самим очищує прапор RXC.

При отриманні даних можуть виникнути помилки, на які вказують прапори регістра UCSRA. Прапор помилки кадру FE встановлюється, якщо стоп-біт прочитаний не коректно. Прапор переповнення DOR встановлюється, якщо буферний регістр UDR містить дані і виявлено новий старт-біт. Прапор помилка парності/непарності PE встановлюється якщо отриманий кадр має таку помилку.

6. Реалізація інтерфейсу USART

Для реалізації інтерфейсу USART потрібно два МК. МК1 програмується для передачі даних, а МК2 – для прийому. Три байти даних для передачі розміщуються в пам’яті програм. Передача починається з натискання кнопки SW4 (Старт) МК1. МК1 в циклі послідовно зчитує байти даних з пам’яті і передає їх через вивід TXD. МК2 послідовно приймає байти даних з виводу RXD і записує в пам’ять SRAM починаючи с адреси $180. Закінчивши прийом МК2, при натисканні кнопки SW5 (Перегляд), послідовно виводить отримані дані на світлодіоди.

Схема з’єднання МК показана на рис. 8. Після ініціалізації обидва МК переходять в режим очікування натисканням кнопки SW4 МК1. МК2 програмується на прийом і його робота залежить від значення прапора RXC: при RXC=0 – очікування, а при RXC=1 – введення байта. Стан прапора RXC залежить від роботи передаючого пристрою. Перехід до прийняття наступного байту відбувається після встановлення прапора закінчення передачі TXC (і відповідно прийому RXC) чергового байту даних. Після закінчення прийому всіх байтів

включаються всі світлодіоди. При наступному послідовному натисканні кнопки SW5 (Перегляд) виводяться на світлодіоди прийняті байти даних.

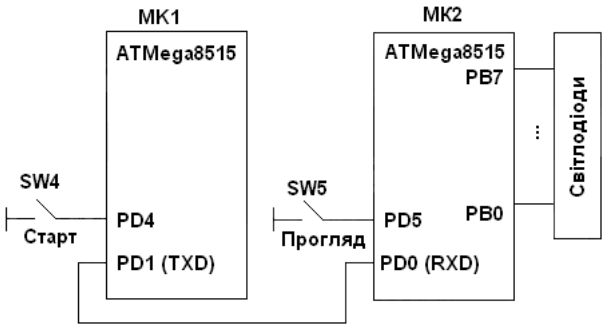


Рис. 8 – З'єднання МК для передачі повідомлень інтерфейсом UART

Алгоритми передачі і прийому даних інтерфейсом USART показані на рис. 9.

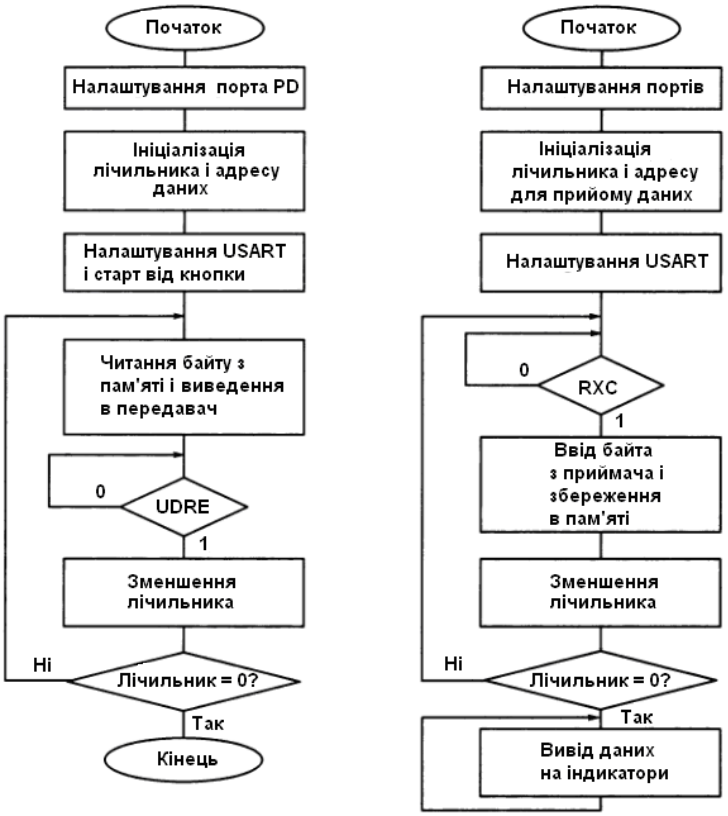


Рис. 9 – Алгоритм передачі і прийому даних інтерфейсом USART

Програмна реалізація інтерфейсу USART показана в прикладах 1, 2. Програми виконуються на двох платах STK500 або в середовищі Proteus, рис. 10.

Приклад 1.

```
;*********************************************************************
;1.asm - робота каналу UART для МК ATx8515
; Натискання кнопки СТАРТ- передача
; по каналу UART трьох байтів, зчитаних з комірок flash-пам'яті.
;Частота тактового генератора F:
; Fint rc = 1МГц, UBRR=12, BR=9600 bprs
; Fint rc = 1МГц, UBRR=2,  BR=19.2 Kprs
```

```
; Fint rc = 4МГц, UBRR=25, BR=9600 bprs
; Fint rc = 4Мгц, UBRR=12, BR=19.2 Kbprs
; Fint rc = 8МГц, UBRR=8,  BR=57.6 Kbprs
;З'єднання: PD4-SW4, GND STK500-1 з GND STK500-2
;***************************************************************
.include "m8515def.inc" ;файл визначень ATmega8515
.def temp = r16         ;тимчасовий регістр
.def count = r17        ;лічильник
.equ START = 4     ;4-й вивід порту PD

.org $000
         rjmp init
;***Ініціалізація МК
INIT:
     ldi ZL,low(text*2) ;завантаження адреси тексту в байтах!!!
          ldi ZH,high(text*2)     ; повідомлення в регістр Z
          ldi count,3             ;встановлення лічильника байтів
          clr temp           ;налаштування
          out DDRD,temp      ; вивода
          ldi temp,0x10      ;  порту PD4
          out PORTD,temp     ;   на введення
 ;***Ініціалізація USART на передавання даних
        clr temp
          out UBRRH,temp
          ldi temp,12         ; UBRR=12, Fosc=4 МГц
          out UBRRL,temp; швидкість передачі 19219 kbps
          ldi temp,(1<<TXEN)  ; дозвіл на передачу
          out UCSRB,temp
          ;ldi temp, (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0)
          ;out UCSRC,temp      ; фрейм передачі: 8-біт даних, 1-стоп біт

WAIT_START:sbic PIND,START  ;очікування натискання кнопки START
        rjmp WAIT_START
 OUTPUT: lpm                ;зчитування байту з flash-пам'яті в r0
 WAIT:  sbis UCSRA,UDRE     ; очікування опорожнення буфера передачі
          rjmp WAIT
          out UDR,r0         ;завантаження даних в буфер передавача
next:     adiw ZL,1     ;збільшення вказівника адресу на 1
          dec count     ;зменшення лічильника на 1
          brne OUTPUT        ;продовження виводу
fin: rjmp fin      ;передача закінчена
text: .db 'A','V','R'   ;текст повідомлення (коди $41,$56,$52)
; 0x41=0b0100_0001, 0x56=0b0101_0110, 0x52=0b0101_0010
```

Приклад 2.

```
;***************************************************************
;2.asm - робота каналу UART для МК ATx8515
;Натискання SW5 - прийом трьох байтів. Fint rc= 4 МГц,
;при UBRR=12 швидкість обміну 19200 bpps
;З'єднання: шлейфом порт PB-LED, PD5-SW5, GND STK500-1 з GND STK500-2
;***************************************************************
.include "m8515def.inc"     ;файл визначень ATmega8515
.def temp = r16             ;тимчасовий регістр
.def count = r17            ;лічильник
.equ SHOW = 5               ;5-й вивід порту PD
```

```
.org $000
         rjmp init
;***Ініціалізація МК
INIT:    ldi temp,low(RAMEND)    ;встановлення
         out SPL,temp       ; вказівника стеку
         ldi temp,high(RAMEND)   ; на останню
         out SPH,temp       ; комірку ОЗП
         ldi YL,0x80             ;в регістрі Y - адрес, по якому
         ldi YH,0x01             ; записуються прийняті дані
         ldi count,3             ;встановлення лічильника байтів
         ser temp           ;налаштування
         out DDRB,temp      ; порту PB на виведення
         out PORTB,temp     ; і виключення світлодіодів
         clr temp
         out DDRD,temp      ;налаштування
         ldi temp,0x20      ; виводу PD5
         out PORTD,temp     ; на введення
;***Ініціалізація UART на приймання даних
         ldi temp,(1<<RXEN) ;дозвіл приймання
         out UCSRB,temp     ; по каналу UART
         ldi temp,12             ;швидкість приймання 19210 кбіт/с
         out UBRRL,temp     ; BRR=12, Fosc=4 MHz
WAIT_RXC:
     sbis UCSRA,RXC     ;очікування отримання даних
         rjmp WAIT_RXC
INPUT:   in temp,UDR             ;введення байту з приймача
         st Y+,temp              ;і збереження в пам'яті
         dec count          ;зменшення лічильника на 1
         brne WAIT_RXC      ;продовження приймання
         clr temp           ;сигналізація –
         out PORTB,temp     ; приймання закінчено
         ldi YL,0x80             ;відновлення
         ldi YH,0x01             ; початкового адресу
         ldi count,3             ;встановлення лічильника байтів
WAIT_SHOW:    sbic PIND,SHOW;очікування натискання
         rjmp WAIT_SHOW     ;  кнопки SW5
         ld temp, Y+             ;зчитування байту з пам'яті
         com temp           ;інвертування
         out PORTB,temp     ;виведення на світлодіоди
         rcall DELAY             ;затримка
         dec count          ;якщо показані не всі дані,
         brne WAIT_SHOW     ; то продовження при натисканні SW5
         ser temp           ;виведення закінчене
         out PORTB,temp     ;світлодіоди виключені
         ldi count,3
         ldi YL,0x80
         ldi YH,0x01
         rjmp WAIT_SHOW          ;повторення виведення

;*** Затримка ***
DELAY:   ldi r19,10
         ldi r20,255
         ldi r21,255
dd:  dec r21
         brne dd
```

```
dec r20
brne dd
dec r19
brne dd
ret
```

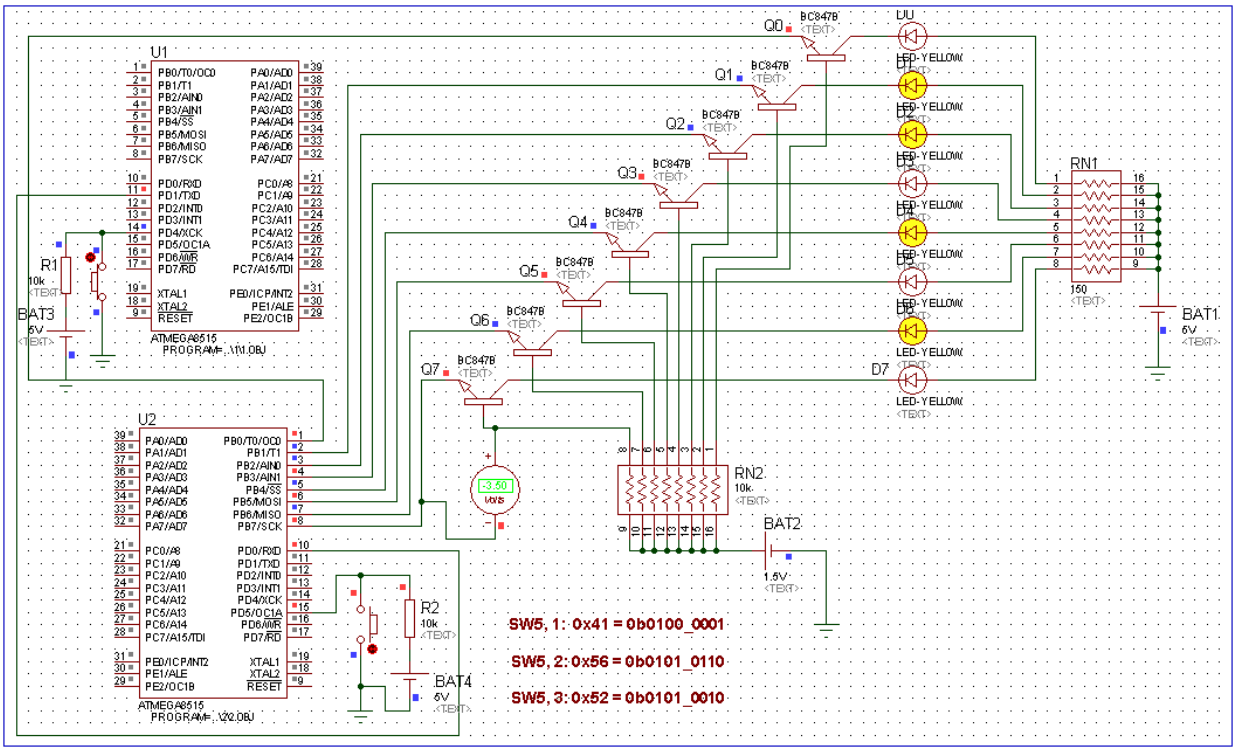


Рис. 10 – Електрична схема з’єднання двох МК для роботи з інтерфейсом USART

Завдання.

1. Написати програму для передачі повідомлення *hello*, яке зберігається в пам’яті програм МК1, в пам’ять даних МК2. Процедуру передачі і процедуру прийому кожного символу повідомлень запрограмувати, використовуючи запити переривань від передавача і приймача.

2. Змінити програму, здійснивши передачу даних з пам’яті SRAM по каналу UART для одного МК. Завантажити дані (константи) в комірки пам’яті SRAM безпосередньо командами програми. Для прийому даних з входу RXD, з’єднавши вихід TXD з входом RXD.

3. Перевірити роботу каналу USART використовуючи програму *Hyper Terminal* для обміну повідомленнями через COM-порт персонального комп’ютера з МК на платі STK 500 через інтерфейс RS-232;

3.1. Створити в AVR Studio 4 проект для прикладу 1. Перевірити роботу програми в режимі симуляції, спостерігаючи стан регістрів і бітів стану каналу USART. Налагоджену програму записати в пам’ять МК на платі STK500. Закрити проект в Atmel Studio 4.

З’єднати при виключеному живленні плату STK500 з комп’ютером. Для цього використати інтерфейс RS-232 для передачі/прийому даних з МК по каналу USART. Для з’єднання використати кабель зв’язку і 9-контактний роз’єм інтерфейсу RS-232, позначений на платі як RS232 SPARE. Підключити МК до інтерфейсу RS-232 дводротовим шнуром. З’єднати виводи порту PD0, PD1 з контактами RXD, TXD двоконтактного роз’єму RS232 SPARE. Після з’єднання подати напругу живлення на плату. Запустити програму *Hyper Terminal* із Windows, вибравши в меню Програми/Стандартні/Зв’язок, і налаштувати канал зв’язку, встановивши

COM-порт, швидкість обміну 19200 бод, 8-бітовий формат. Натиснути кнопку Виклик. На платі замкнути кнопку SW4 (Старт) для запуску передачі повідомлення. Продивитися у вікні програми *Hyper Terminal* комп'ютера прийняте повідомлення. Повторити передачу, змінивши швидкість передачі даних. Перевіривши роботу каналу, закрити програму *Hyper Terminal*.

3.2. Створити в Microchip Studio проект для прикладу 2. Перевірити роботу програми в режимі симуляції, вручну встановлюючи прапор прийому RXC і дані в регістрі UDR і контролюючи вивід даних у вихідний порт PB. Налагоджену програму записати в пам'ять МК на платі STK500. Закрити проект.

Підготовити текстовий файл короткого повідомлення (з трьох символів) з використанням стандартної програми Блокнот. Виконати з'єднання, як в п.3.1. Включити напругу живлення на платі. Знову запустити програму *Hyper Terminal* і налаштувати канал зв'язку (швидкість – 19200 біт/с, біти даних – 8, парність – ні, стопові біти – 1, керування потоком – Xon/Xoff). Виконати команду меню Передача/Відправити текстовий файл. Після включення світлодіодів на платі вивести на індикатори ASCII-коди прийнятих символів, тричі натискаючи кнопку SW5. Виконавши скидування МК, повторити передачу потоку цифрових даних безпосередньо з клавіатури.

4. Перевірити роботу каналу USART при передачі даних між МК1 і МК2. З'єднати напряму вивід PDl (TXD) першого STK500 з виводом PD0 другого STK500, а також з'єднати виводи GND обох плат. Плати STK500 після програмування від'єднати від кабелів RS-232. Натисканням кнопок RESET обидва МК перевести в режим очікування. Передача повідомлення починається після натискання кнопки SW4 (Старт) на платі першого STK500. Після прийому повідомлення загоряються всі світлодіоди другого STK500, показуючи, що прийом даних закінчено. Отримані дані можна вивести на індикатори, натискаючи послідовно кнопку SW5.

Примітка. Номер варіанта завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Текст програми із поясненнями.
4. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми

Питання.

1. В чому відмінність інтерфейсу USART від інтерфейсів SPI, I2C.
2. Які регістри входять в склад USART і їх призначення?
3. Який формат кадру даних, які передаються?
4. Яке призначення регістра UDR?
5. Яке призначення регістрів UCSRA, UCSRB, UCSRC?
6. Яке призначення регістра UBRR?
7. Як задати швидкість обміну для USART?
8. Як здійснюється ініціалізація МК перед роботою USART?
9. Як здійснюється передача даних в USART?
10. Як здійснюється прийом даних в USART?

ЛАБОРАТОРНА РОБОТА № 10. Послідовний обмін даними по каналу SPI

Мета роботи – вивчення послідовного обміну даними через канал SPI (Serial Peripheral Interface) та програмування введення/виведення.

1. Інтерфейс SPI

Інтерфейсі SPI (Serial peripheral interface, послідовний периферійний інтерфейс) підтримує високошвидкісну синхронну передачу даних між МК та периферійними пристроями або між декількома МК. Інтерфейс SPI в Atmega8515 має наступні особливості:

- повнодуплексна, 3-дротова синхронна передача даних;
- режим робота Ведучий (Master) або підпорядкований (Slave);
- передача даних починаючи з молодшого або старшого біту;
- прапор переривання кінець передачі;
- прапор захисту від колізій при запису даних;
- вихід з режиму очікування Idle;
- режим роботи ведучого SPI з подвоєною швидкістю.

У інтерфейсі SPI дані передаються по одній лінії побітово з визначеними інтервалами часу, а синхронізуючі імпульси – по окремій виділеній лінії. Це спрощує задачу синхронізації – не потрібно спеціально задавати швидкості обміну, але при цьому збільшується число сигнальних ліній (не менше трьох). Для адресації, при підключенні більше ніж двох МК до одного інтерфейсу, потрібний додатковий четвертий вивід /SS (Slave select). Він позначається з інверсією, так як вибір здійснюється подачею низького рівня на цей вхід.

В склад модуля SPI входять (рис. 1):

– 8-розрядний зсувний регістр SPDR (приймає байт з шини даних МК, зсовує його вправо або вліво з видачею послідовного коду на вивід МК, одночасно з виводом приймає послідовний код із входу МК і через буферний регістр передає його на шину даних МК);

– 8-розрядні регістри керування SPCR і стану SPSR;

– попередній подільник частоти;

– схеми керування.

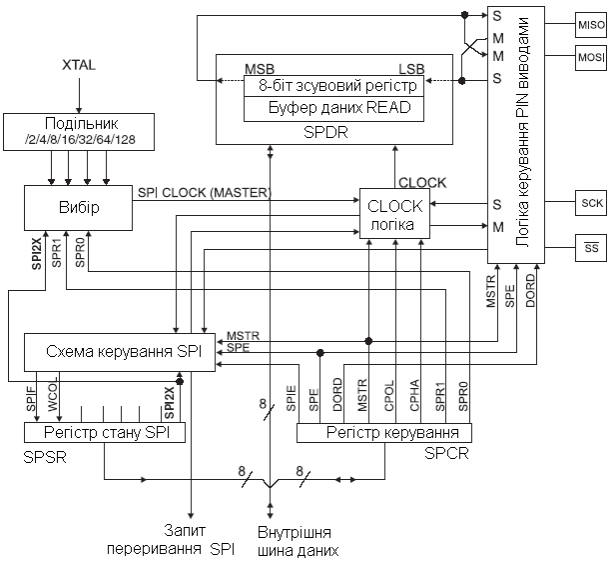


Рисунок 1 – Структурна схема SPI

В МК ATmega8515 для сигналів інтерфейсу SPI призначені наступні виводи порту PB: PB5 – MOSI, PB6 – MISO, PB7 – SCK, PB4 – SS.

Регістр керування SPCR задає режими роботи SPI:

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

• SPIE (SP interrupt enable) – дозвіл переривання SPI (якщо встановлені біти SPSR.SPIF і SREG.I);

• SPE (SP enable) – дозвіл доступу до SPI (якщо біт встановлений);

• DORD (Data order) – порядок даних (1 – першим передається LSB біт, 0 – першим передається MSB біт);

• MSTR (Master/Slave select) – режим роботи порту SPI (1 – ведучий, 0 – ведений);

• CPOL (Clock Polarity) – полярність тактового сигналу (в режимі очікування тактовий сигнал SCK має низький логічний рівень – 0 або високий логічний рівень – 1);

Функціонування CPOL:

CPOL	Ведучий фронт	Завершальний фронт
0	Наростання	Спадання
1	Спадання	Наростання

• CPHA (Clock Phase) – вибір фази тактового сигналу (читання біта по ведучому фронту сигналу або завершальному фронту тактового сигналу:

CPHA	Ведучий фронт	Завершальний фронт
0	Вибірка	Встановлення
1	Встановлення	Вибірка

• SPR1, SPR0 (SP Clock Rate Select 1 and 0) – вибір частоти тактового сигналу ведучого МК.

SPI2X	SPR1	SPR0	Тактова частота
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Регістр стану SPI (SPI Status Registr) – **SPSR**

7	6	5	4	3	2	1	0
SPIF	WCOL	-	-	-	-	-	SPI2X

• SPIF (SPI Interrupt flag) – прапор виставляється при закінченні передачі.

• WCOL (Write COLlision Flag) – прапор встановлюється, якщо під час передачі є спроба запису даних.

• SPI2X (Double SPI Speed Bit) – коли прапор встановлюється в 1 тактова частота подвоюється для для ведучого МК.

Регістр даних SPI (SPI Data Registr) – **SPDR**:

7	6	5	4	3	2	1	0
MSB							LSB

В регістр SPDR дані записуються і дані з нього зчитуються при передачі даних між регістровим файлом і зсувним регістром SPI (SPI shift register). Запис даних у регістр починає передачу. Читання регістра спричиняє читання отримуючого буфера зсувного регістру.

В залежності від комбінації бітів CPHA і CPOL є чотири формати передачі даних:

	Ведучий фронт	Завершальний фронт	SPI режим
CPOL=0,CPHA=0	Вибірка (Наростання)	Встановлення (Спадання)	0
CPOL=0,CPHA=1	Встановлення (Наростання)	Вибірка (Спадання)	1
CPOL=1,CPHA=0	Вибірка (Спадання)	Встановлення (Наростання)	2
CPOL=1,CPHA=1	Встановлення (Спадання)	Вибірка (Наростання)	3

При MSTR=1 порт SPI працює як ведучий. При цьому вивід MOSI є виходом даних, вивід MISO – входом даних, вивід SCK – виходом для імпульсів, які використовуються як зсувні при прийманні даних ведучим МК. Функція виводу SS залежить від стану розряду DDRB.4. При DDRB.4=1 вивід SS порту SPI не підключений до виводу порту PB4, при DDRB.4=0 значення сигналу на вході впливає на роботу порту SPI. Якщо PB4(SS)=1, порт працює як ведучий, а при значенні сигналу 0 перемикається в режим веденого. Для переведення порту SPI в робочий стан встановлюється біт SPE регістра SPCR. При MSTR=0 порт SPI працює в режимі веденого. При цьому вивід MOSI є входом даних, вивід MISO – виходом даних, вивід SCK – входом для імпульсів зсуву, вивід SS – входом. Виводи SPI підключаються до виводів порту PB також при встановленні біта SPE регістра SPCR. Порт переходить в робочий стан за сигналом логічного 0 на вході SS. Схема з'єднання двох МК для обміну даними по каналу SPI зображена на рис. 2.

Передача даних починається після запису даних у регістр SPDR ведучого МК. Порядок видачі визначається станом біта DORD. Після видачі останнього розряду встановлюється прапор SPIF (біт 7 регістра SPSR) і одночасно виробляється запит переривання SPI_STC з адресою вектора $008 (при встановленому прапорі SPIE регістра SPCR). Одночасно з передачею приймаються дані від веденого МК. По закінченню прийому даних у веденому МК також встановлюється в 1 прапор SPIF і виробляється запит на переривання. При спробі запису у

регістр даних SPDR під час передачі чергового байту встановлюється в 1 прапор WCOL регістра SPSR. Швидкість передачі ведучого МК задається бітами SPR1, SPR0 регістра SPCR.

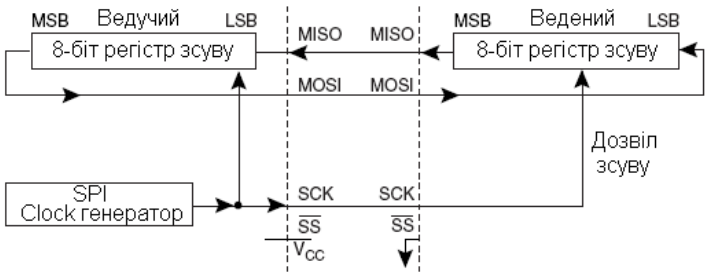


Рисунок 2 – Схема з’єднання двох МК

2. Приклади ініціалізації SPI як ведучого і веденого

Приклади коду для ініціалізації SPI як Master і як виконати просту передачу. DDR_SPI у прикладах потрібно замінити на фактичний регістр напрямку, який контролює вивід SPI. DD_MOSI, DD_MISO та DD_SCK потрібно замінити фактичними бітами напрямків передачі даних для цих виводів. Наприклад, якщо MOSI розміщено на виводі PB5, то замінити DD_MOSI на DDB5, а DDR_SPI на DDRB.

```
SPI_MasterInit:
; Set MOSI and SCK output, all others input
ldi r17,(1<<DD_MOSI)|(1<<DD_SCK)
out DDR_SPI,r17
; Enable SPI, Master, set clock rate fck/16
ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
out SPCR,r17
ret

SPI_MasterTransmit:
; Start transmission of data (r16)
out SPDR,r16
Wait_Transmit:
; Wait for transmission complete
sbis SPSR,SPIF
rjmp Wait_Transmit
ret
```

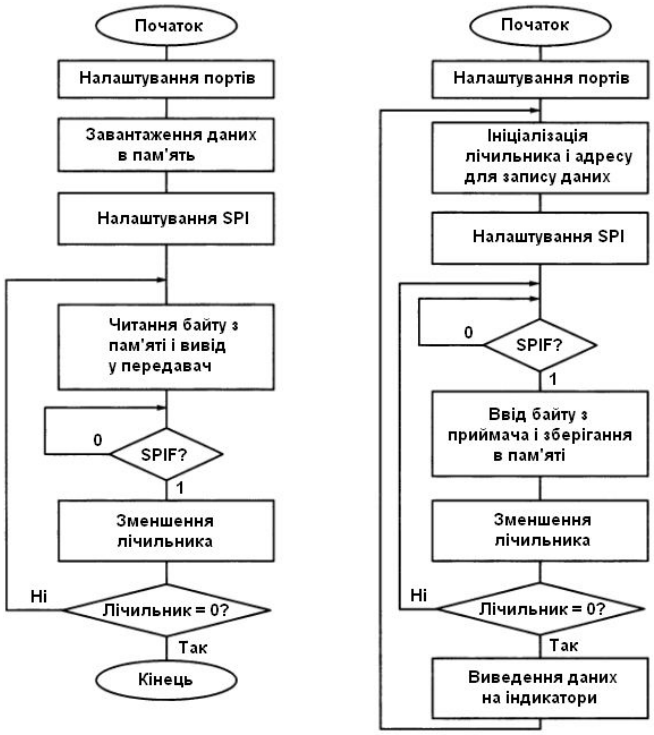
Наступний коду показує, як ініціалізувати SPI як підпорядкований і як виконати просте приймання даних.

```
SPI_SlaveInit:
; Set MISO output, all others input
ldi r17,(1<<DD_MISO)
out DDR_SPI,r17
; Enable SPI
ldi r17,(1<<SPE)
out SPCR,r17
ret

SPI_SlaveReceive:
```

```
; Wait for reception complete
sbis SPSR,SPIF
rjmp SPI_SlaveReceive
; Read received data and return
in r16,SPDR
ret
```

3. Реалізація інтерфейсу SPI між двома МК

Для реалізації інтерфейсу SPI потрібно два МК. МК1 передає дані, а МК2 – приймає. Виходом передавача МК1 є вивід MOSI (PB5), а входом приймача МК2 – вивід MOSI (PB5). Дані для передачі (три байти) завантажуються в комірку пам'яті SRAM, починаючи з адреси 0x170 ($170). Потім в циклі дані послідовно виводяться. Перехід до наступної ітерації в циклах здійснюється після встановлення прапора SPIF. Після прийняття даних в МК2, вони записуються в його пам'ять SRAM починаючи з адреси 0x180 ($180). Після прийняття усіх даних, при натисканні кнопки SW5, вони послідовно виводяться на світлодіоди. Схеми алгоритмів передачі і прийому даних інтерфейсом SPI показані на рис. 3. Програмна реалізація інтерфейсу SPI показана в прикладах 1, 2. Програми виконуються на двох платах STK500 або в середовищі Proteus, рис. 4.



а *б*

Рисунок 3 – Схеми алгоритмів передачі даних з МК1 (*а*) і прийому в МК2 (*б*) по каналу SPI

Приклад 1.

```
;*****************************************************************
;1.asm - робота каналу SPI, режим МК1 - master
;Після скидання  МК1 передаються три байти,
;які зчитуються з комірок SRAM за адресами з регістра Z.
;З'єднання: PB5(МК1)-PB5(МК2), PB7(МК1)-PB7(МК2), PB0(МК1)-PB4(МК2)
;*****************************************************************
```

```
;.include "m8515def.inc"    ;файл визначень ATmega8515
.def temp = r16         ;тимчасовий регістр
.def count = r17        ;лічильник
.org $000
         rjmp init
;***Ініціалізація МК
INIT:    ldi temp,0b1011_0001 ; 0x51, SCK,MOSI,SS, PB0
         out DDRB,temp ;     на вивід
         ldi ZL,0x70         ;завантаження адресу
         ldi ZH,0x01
         ldi temp,0x0f ; завантаження даних
         st Z+,temp          ; з використанням
         ldi temp,0x3f ; непрямої
         st Z+,temp          ; адресації і
         ldi temp,0xff ; постінкременту
         st Z+,temp
         ldi count,0x03;встановлення лічильника передач
;***Налаштування SPI в режимі MASTER на передачу даних
         ldi temp,(1<<SPE)|(1<<MSTR)
         out SPCR,temp
OUTPUT:  ldi temp,0b1000_0001   ;переключення
         out PORTB,temp     ;сигналу на виході PB0
         ldi temp,0b1000_0000    ; з 1
         out PORTB,temp     ; в 0
         ld temp,-Z              ;зчитування байту з пам'яті
         out SPDR,temp      ;вивід байту в передавач
Wait_Transmit:
         sbis SPSR,SPIF     ; перевірка прапора передачі
         rjmp Wait_Transmit
         dec count          ;зменшення лічильника на 1
         brne OUTPUT
loop:    rjmp loop

;*********************************************************************
;2.asm - робота каналу SPI, режим МК2 - SLAVE
;Після скидання  МК2 приймаються три байти, які записуються в SRAM
;за адресами з регістру X.
;По закінченню прийому засвічуються всі світлодіоди.
;При послідовному натисканні на SW5 (SHOW), дані
;зчитуються з пам'яті і виводяться на світлодіоди.
;З'єднання: SW5-PD5, шлейфом порт PC-LED
;*********************************************************************
;.include "m8515def.inc"    ;файл визначень ATmega8515
.def temp = r16             ;тимчасовий регістр
.def count = r17        ;лічильник
.equ SHOW = 5           ;5-й вивід порту PD
.org $000
         rjmp init
;***Ініціалізація МК
INIT:
         ldi temp,low(RAMEND)    ;встановлення
         out SPL,temp       ; вказівника стеку
         ldi temp,high(RAMEND)   ;  на останню
         out SPH,temp       ;   комірку ОЗП
         ldi temp,0b0100_0000  ;  0x40 (0100_0000), PB6,MISO на передачу
```

```
          out DDRB,temp          ; PB5,MOSI на прийом
          ldi temp,0b1011_0000   ; 0xb0, підтягуючі резистори
          out PORTB,temp
          clr temp           ;налаштування
          out DDRD,temp      ; виводу
          ldi temp,0x20      ;  порту PD5
          out PORTD,temp     ;   на введення (0010_0000)
          ldi temp,0xff      ;налаштування
          out DDRC,temp      ; виводів порту PC
          out PORTC,temp     ;  на вивід
          ldi count,0x03     ;встановлення лічильника байтів
          ldi XL,0x80             ;в регістрі X адрес, по якому
          ldi XH,0x01             ; відбувається запис прийнятих даних
;***Налаштування SPI в режимі SLAVE на прийом даних
          ldi temp,(1<<SPE)
          out SPCR,temp
INPUT:    sbis SPSR,SPIF;перевірка прапора прийому
          rjmp INPUT
          in temp,SPDR  ;введення байту з приймача
          st X+,temp         ;збереження байту в пам'яті
          dec count     ;зменшення лічильника на 1
          brne INPUT
          clr temp      ;сигналізація -
          out PORTC,temp; приймання закінчено
          ldi count,3        ;встановлення лічильника байтів
WAIT_SHOW:    sbic PIND,SHOW;очікування натискання
          rjmp WAIT_SHOW; кнопки SHOW
          ld temp, -X        ;зчитування байту з пам'яті
          ;com temp     ;інвертування і
          out PORTC,temp;виведення на світлодіоди
          rcall DELAY        ;затримка
          dec count     ;якщо показані не всі дані,
          brne WAIT_SHOW; то продовження по натисканню SHOW
          rjmp INPUT
;***Затримка***
DELAY:    ldi r19,5
          ldi r20,255
          ldi r21,255
dd:  dec r21
          brne dd
          dec r20
          brne dd
          dec r19
          brne dd
          ret
```

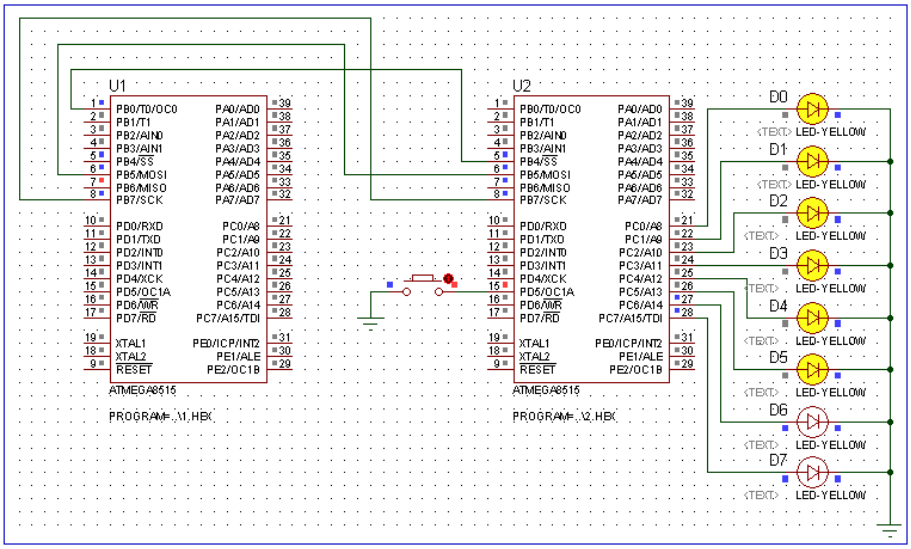


Рисунок 4 – Електрична схема з’єднання двох МК для роботи з інтерфейсом SPI

Завдання.

1. Дослідити роботу двох МК з інтерфейсом SPI у середовищі Proteus (рис. 3). Запустити режим налагодження Debug/Start restart debugging. Виділити правою кнопкою миші МК1 і у виборі AVR позначками CPU register, Data memory, I/O register, Source code відкрити відповідні вікна. Аналогічні вікна відкрити для МК2. В покроковому режимі Step over (F10) показати з яких комірок пам’яті даних МК1 і в які комірки пам’яті даних МК2 передаються дані.

2. Змінити програму 1.asm, задавши разове перемикання сигналу з 1 в 0 на виводі PB0 перед виводом першого байту і зберігати рівень 0 для всіх наступних передач. Показати шляхом моделювання в Proteus, як ці зміни впливають на роботу інтерфейсу.

3. Змінити програми 1.asm, 2.asm, задавши CPOL=1. Промоделювати роботу інтерфейсу в Proteus і пояснити, як ці зміни впливають на полярність сигналів лінії SCK (PB7).

4. Змінити програми 1.asm, 2.asm, задавши DORD=1. Промоделювати роботу інтерфейсу в Proteus і пояснити, як ці зміни впливають на порядок виводу бітів даних на лінії MOSI.

5. Замінити в програмах 1.asm, 2.asm програмну перевірку прапора готовності SPIF обробкою запитів переривання SPI_STC. Після початку передачі перевести МК1 в режим очікування, перериваючи його за запитами переривань. МК2 після ініціалізації також ввести в режим очікування.

Примітка. Номер варіанта завдання вибирається за порядковим номером студента у журналі групи.

Звіт з лабораторної роботи має містити:

1. Номер групи, ПІБ студента, завдання до роботи.
2. Короткий опис теоретичної частини.
3. Текст програми із поясненнями.
4. Роздруки екранів електричної схеми в Proteus/Isis з результатами виконання програми

Питання.

1. Як передаються дані в інтерфейсі SPI?
2. Розказати про структурну схему інтерфейсу SPI?
3. Розказати про регістр стану SPSR і регістр даних SPDR.

4. Розказати про регістр керування SPCR і його розряди.
5. Як задати частоту тактового сигналу ведучого МК?
6. Як організувати в SPI режим ведучий і ведений.

ЛАБОРАТОРНА РОБОТА № 11. Послідовний обмін даними по каналу TWI

Мета роботи – вивчення прийому і передачі даних по двопровідному послідовному інтерфейсу TWI.

1. Інтерфейс TWI

В деяких моделях МК Atmega є двопровідний послідовний байт орієнтований інтерфейс TWI (Two-wire Serial Interface). Інтерфейс TWI має наступні особливості:

- для передачі даних використовується тільки два проводи;
- підтримуються операції Ведучий (Master) – Ведений (Slave);
- пристрій може функціонувати як передавач або приймач;
- 7-бітовий адресний простір дозволяє підключення до 127 ведених (slave) пристроїв;
- підтримується арбітраж багатьох ведучих (master) пристроїв;
- швидкість передачі даних до 400 kHz;
- драйвери з обмеженою вихідною швидкістю;
- схема подавлення завад відкидає імпульсні сплески сигналів на лініях шин;
- програмовані адреси підпорядкованих пристроїв із підтримкою загального виклику.
- розпізнавання адрес викликає пробудження МК із режиму сну.

Двопровідний послідовний інтерфейс TWI забезпечує взаємодію МК з набором інших мікросхем (енергонезалежною пам’яттю, контролерами паралельних портів, LCD-дисплеями, іншими МК). Інтерфейс TWI дозволяє об’єднати до 128 пристроїв, використовуючи дві двонаправлені лінії, одну SDA для даних, другу SCL для тактових сигналів.

Всі пристрої під’єднані до інтерфейсу TWI мають індивідуальні адреси. Пристрій, який починає і закінчує передачу називається ведучим (Master). Ведучий пристрій також генерує тактові сигнали. Ведений пристрій адресується ведучим пристроєм. Статус пристрою задається програмно.

1.1. Електричне з’єднання

Обидві лінії шини через зовнішні підтягуючі резистори R1, R2 підключені до джерела живлення VC, рис. 1. Драйвери всіх пристроїв виконані по схемі з відкритим колектором або відкритим стоком, що дозволяє реалізувати “монтажне І” для вихідних сигналів. Низький рівень на виході одного або декількох пристроїв встановлює лінію шини в низький рівень. Високий рівень встановлюється на лінії шини, коли виходи всіх пристроїв знаходяться у високоімпедансному (третьому) стані, дозволяючи підтягуючим резистором підтягнути лінію до високого рівня. Пристрої підключені до TWI повинні мати живлення, для виконання любих операцій з шиною.

Число підключених до шини пристроїв обмежується ємністю шини в 400 пФ і 7-бітовим адресним простором.

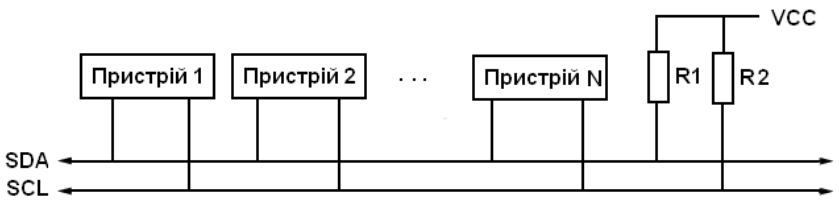


Рисунок 1 – Схема з’єднання пристроїв в інтерфейсі TWI

2. Передача даних і формат пакету

2.1. Передача біту

Кожний біт, який передається по лінії даних SDA супроводжується імпульсом на тактовій лінії SCL, рис. 2.

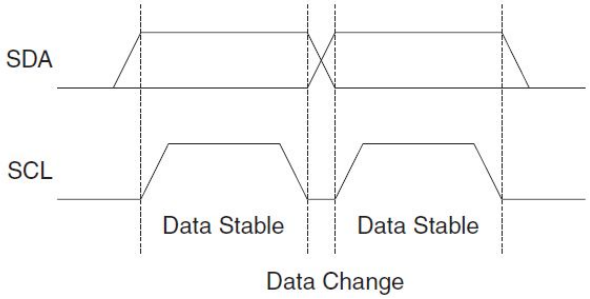


Рисунок 2 – Передача біту даних

2.2. Умови початку (START) і зупинки (STOP) передачі даних

Інтерфейс TWI є послідовним: всі дані і адреси передаються по лінії SDA побітово. Ведучий пристрій починає передачу станом START, а закінчує – станом STOP. Стартовий біт формується шляхом зміни рівня сигналу на лінії SDA 1=>0 при SCL лог.1, а стоповий – при зміні рівня сигналу на лінії SDA 0=>1 при SCL лог.1, рис. 3. Кожний біт, який передається супроводжується тактовим сигналом на лінії SCL, який формує ведучий пристрій. За час дії сигналу SCL (лог.1) стан лінії SDA має залишатися незмінним. Між станами START і STOP шина вважається зайнятою і ніякий інший ведучий пристрій не може отримати контроль над шиною. Особливий випадок трапляється, коли між умовами СТАРТ та СТОП видається нова умова START. Це називається умовою REPEATED START і використовується, коли ведучий бажає ініціювати нову передачу без відмови від управління шиною. Після REPEATED START шина вважається зайнятою до наступного стану СТОП. Це ідентично до поведінки START, і тому START використовується для опису як START, так і REPEATED START.

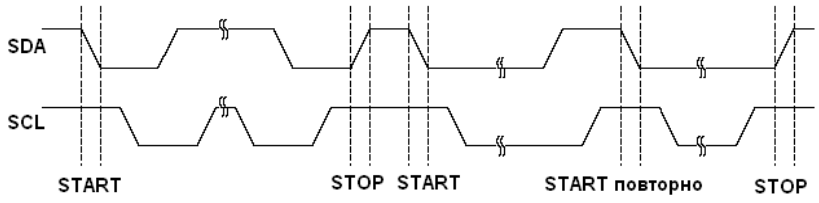


Рисунок 3 – Умови START, REPEATED START, STOP

2.2. Формат адресних пакетів і пакетів даних

По шині можуть передаватися два типи пакетів: адресні і з даними.

Адресні пакети. Всі адресні пакети є 9-бітовими і містять: 7-бітовий адрес, який передається веденим пристроям (починаючи із старшого), контрольний біт читання/запису R/W(1 – читати R, 0 – писати W), біт підтвердження ACK, рис. 4.

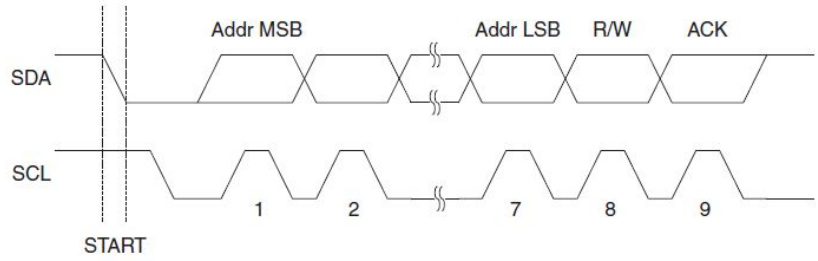


Рисунок 4 – Формат адресного пакету

Адрес 0000_000 використовується для загального виклику. Адреси 1111_xxx зарезервовані. У відповідь на загальний виклик всі підпорядковані пристрої переводять лінію SDA у лог.0 у ACK-циклі підтвердження (9-й такт SCL). Кадр даних, який після цього передається, буде отриманий всіма підпорядкованими пристроями, які встановили підтвердження. Для виклику одного підпорядкованого пристрою задається його адрес у кадрі адресу. Кожний із пристроїв порівнює поступаючий адрес із власним. При розпізнанні підпорядкованим пристроєм свого адресу, він повертає у лінію SDA сигнал підтвердження АСК (лог.0 SDA на 9-такті SCL). Після отримання підтвердження ведучий пристрій починає передачу кадра даних.

Пакет даних. Пакет даних містить 8-бітів даних і один біт підтвердження ACK, який формує приймач, рис. 4.

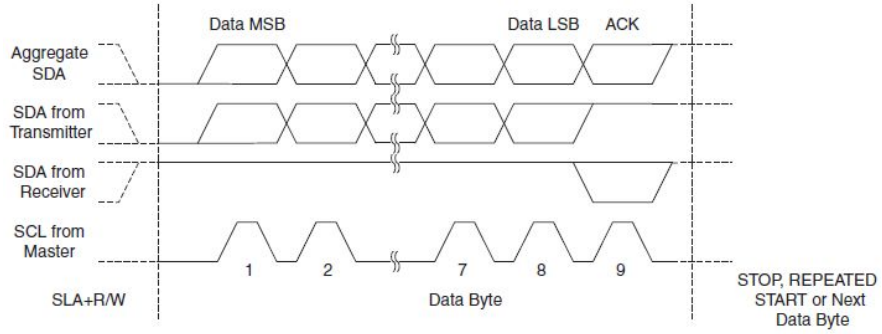


Рисунок 5 – Формат пакету даних

Дані передаються послідовно, починаючи із старшого біта. Після прийому кожного байта приймач виробляє сигнал підтвердження АСК. Не отримавши підтвердження від приймача, передавач може зупинити передачу даних, сформувавши стан STOP.

Приймач може збільшити тривалість періоду SCL у лог.0. Така зміна не впливає на тривалість SCL у лог.1, яку формує передавач. Збільшення тривалості періоду SCL у лог. 0 дозволяє вирівняти швидкодію передавача і приймача.

Після закінчення передачі байту даних, можуть передаватися наступні байти без зміни приймача, вибиратися новий приймач або закінчуватися обмін із звільненням шини.

3. Структура модуля TWI

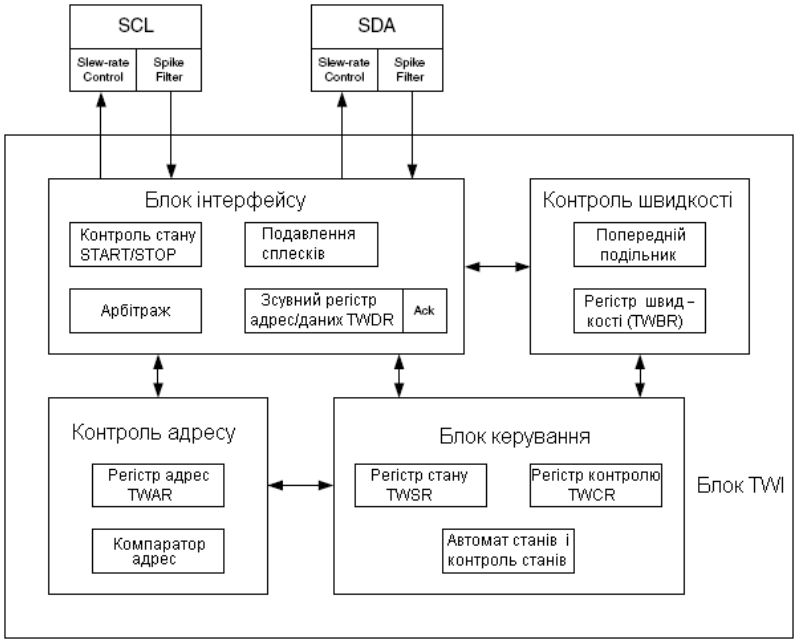


Рисунок 6 – Структура модуля TWI

Модуль містить блок шинного інтерфейсу (SDA, SCL) з регістром даних TWDR і контролером станів Start/Stop, блок контролю адреса з регістром TWAR і схемою порівняння, блок керування з регістрами TWCR і TWSR, контролер швидкості передачі з попереднім дільником і регістром швидкості TWBR. Виводи SCL, SDA з'єднують TWI з іншими блоками МК. Вихідний драйвер обмежує швидкість, а вхідний – фільтрує імпульси з тривалістю менше 50 нс. Блок контролю швидкості задає частоту імпульсів на лінії SCL в режимі роботи ведучого. Частота імпульсів задається значеннями бітів в регістрах TWBR і TWSR. Частота тактових сигналів МК, працюючого в режимі веденого, має бути в 16 більшою від частоти SCL. SCL частота визначається наступним рівнянням:

$$SCL = \frac{CPU_{CLOCK}}{16 + 2},$$

де *TWPS* значення бітів попереднього дільника в регістрі TWSR.

Блок інтерфейсу містить зсувний регістр даних/адрес TWDR, START/STOP контролер і апаратний арбітраж. В регістрі TWDR зберігаються дані/адреси, які передаються або приймаються і біт підтвердження прийому/передачі. START/STOP контролер генерує і визначає умови START, повторний START і STOP.

Блок контролю адресу перевіряє чи отриманий адрес співпадає з 7-розрядним адресом в регістрі TWAR. Блок контролю порівнює адреси навіть якщо МК знаходиться в режимі sleep.

Блок керування аналізує зміни стану шини згідно із станом регістру TWCR і генерує відповідь, яка змінює стан регістру TWSR або генерує TWI переривання TWINT.

Модуль TWI може працювати в наступних режимах: ведучий з передачею байтів, ведучий з прийомом байтів, ведений з прийомом байтів, ведений з передачею байтів.

4. Регістри TWI

TWBR (TWI Bit Rate Register) – регістр темпу передачі бітів дозволяє вибрати коефіцієнти ділення для генератора темпу передачі бітів. Генератор темпу передачі бітів є частотним дільником, який задає частоту тактових сигналів SCL у режимі ведучого МК.

7	6	5	4	3	2	1	0
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0

TWCR (TWI Control Register) – регістр дозволяє контролювати операції TWI.

7	6	5	4	3	2	1	0
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE

TWINT (TWI Interrup Flag) – прапор встановлюється апаратно, коли TWI завершив роботи і очікує відповіді програми.

TWEA (TWI Enable Acknowledge Bit) – контролює генерацію імпульсу підтвердження (TWEA=1). При TWEA=0 пристрій тимчасово від’єднується від шини.

TWSTA (TWI START Condition Bit) – пристрій записує в TWSTA=1, хоче стати ведучим. Апаратура перевіряє чи шина доступна і вільна та генерує умову START.

TWSTO (TWI STOP Condition Bit) – при запису TWSTO=1 у режимі Ведучий, генерується умова STOP, якщо шина вільна.

TWWC (TWI Write Collision Flag) – біт встановлюється в лог. 1 при спробі запису у регістр TWDR коли TWINT=0. Прапор очищається при запису в TWDR, коли TWINT=1.

TWEN (TWI Enable Bit) – при TWEN=0, TWI отримує контроль над виводами МК, які під’єднані до SCL і SDA, що дозволяє низькошвидкісну передачу і використання фільтра імпульсних завад.

TWIN (TWI Interrupt Enable) – при TWIN=1 і SREG.I=1 активується запит на TWI переривання, якщо прапор TWINT=1.

TWSR (TWI Status Register) – регістр відображає стан TWI логіки і шини (TWS7-TWS3) та дозволяє задати коефіцієнти дільника генератора темпу бітів.

7	6	5	4	3	2	1	0
TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

TWPS1	TWPS0	Значення
0	0	1
0	1	4
1	0	16
1	1	64

TWDR (TWI Data Register) – регістр містить наступний байт для передачі, а в режимі приймання – останній прийнятий байт.

7	6	5	4	3	2	1	0
TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0

TWAR (TWI Address Register) – регістр містить в старших 7-бітах адрес підпорядкованого пристрою в режимах Slave Transmitter або Receiver. В режимі Master не використовується. Біт TWGCE використовується для розпізнання адресу загального виклику (0x00).

7	6	5	4	3	2	1	0
TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE

5. Алгоритми основних транзакцій інтерфейсу TWI

В МК Atmel протокол TWI можна реалізувати програмно або програмно-апаратно у тих моделях, які мають вбудований апаратний протокол TWI. Для програмної реалізації інтерфейсу TWI потрібний набір програм, які емулюють роботу ведучого і веденого пристрою. Набір програм зменшується, якщо ведений пристрій має вбудований порт із інтерфейсом TWI. Алгоритми основних транзакцій шини, запису і читання, показані на рис. 7, 8.

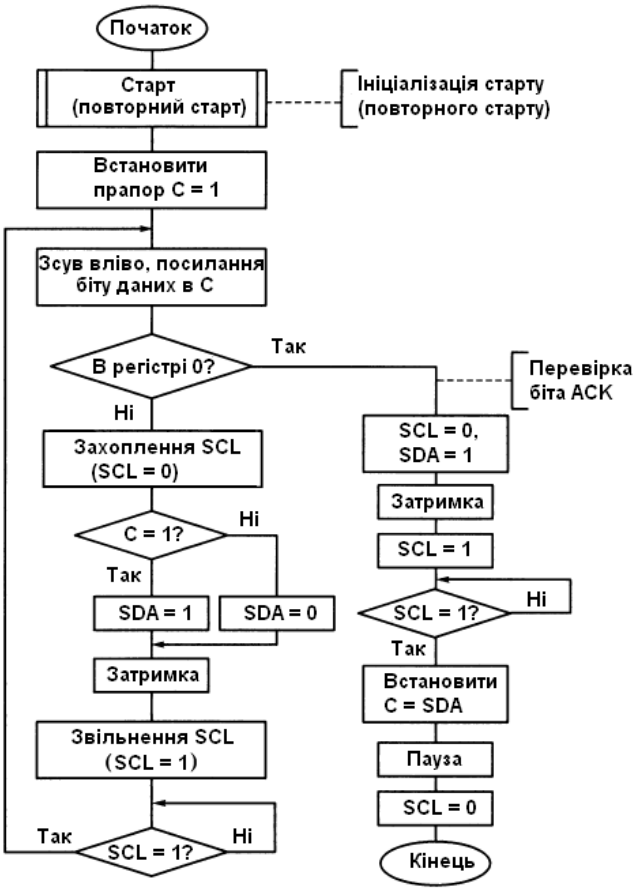


Рисунок 7 – Схема алгоритму запису байту даних

Процедура транзакції запису (передачі адресного байту і запис байту даних) починається із захоплення лінії SDA (SDA = 0) – формування стартового біту і встановлення прапора C = 1, який використовується для виявлення признаку закінчення циклу передачі. Шляхом циклічного зсуву вліво байта даних перший біт, який передається, витісняє 1 з прапора C в молодший розряд регістра даних. Це запобігає можливості передчасного виходу з циклу, коли в регістрі даних під час передачі байта залишаються нульові біти. Лінія SCL переводиться в лог. 0. Значення біту C використовується для керування станом лінії SDA. Якщо передаваний біт, встановлений в C, дорівнює 1, лінія SDA приймає значення SDA = 0, у іншому випадку SDA = 1. Далі через час затримки встановлюється лінія SCL в лог. 1 і після перевірки, якщо ведений пристрій не гальмує роботу на лінії SCL, виконується циклічний перехід для виведення наступного біту даних. На наступних ітераціях циклу виконується логічний зсув. Після виявлення признаку кінця передачі, коли всі біти регістра даних дорівнюють 0, виконується перехід до процедури перевірки біта підтвердження.

Отримання біта підтвердження ACK починається із захоплення лінії SCL (SCL = 0) і звільнення ведучим лінії SDA (SDA = 1). Після часової паузи лінія SCL перемикається в стан 1 і, якщо вона вільна від впливу ведених пристроїв, значення SDA зчитується як сигнал підтвердження ACK (встановлення або скидання біту C). Після паузи на лінії SCL встановлюється 0.

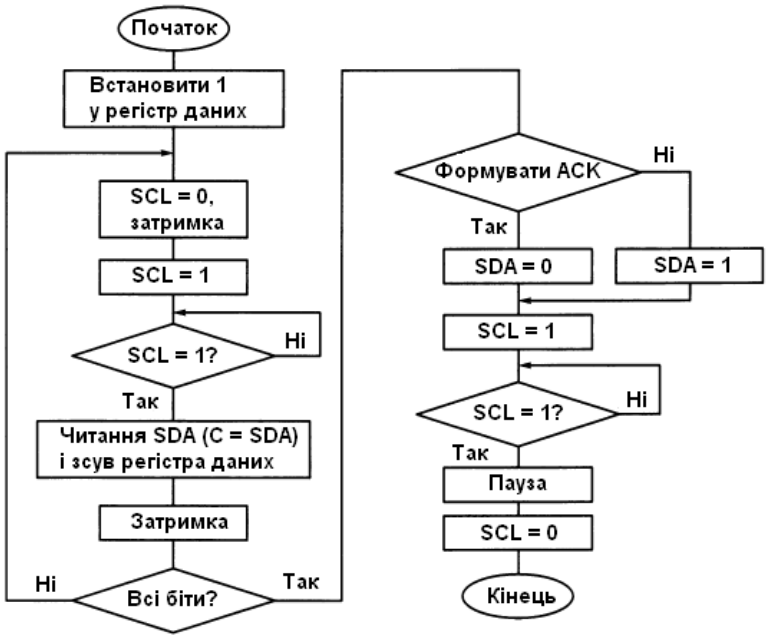


Рисунок 8 – Схема алгоритму читання байту даних

Процедура транзакції читання (прийом даних від веденого) починається з встановлення признаку кінця прийому (1) у молодший розряд регістра даних. Цикл читання бітів даних починається із захоплення лінії SCL=0, встановлення на лінії SCL=1, підтвердження високого рівня сигналу SCL=1 і наступного введення біту даних в регістр даних шляхом опитування лінії SDA і зсуву вмісту регістра даних. Після чергової паузи виконується перевірка признаку кінця прийому байта даних за значенням прапора C. Якщо не всі біти отримані (C=0), прийом продовжується. Якщо прийняті всі біти (C=1), ведучий пристрій переходить до формування біта підтвердження прийому ACK для веденого пристрою. При необхідності формування біта ACK лінія SDA встановлюється в 0, лінія SCL переводиться в стан 1. Після підтвердження SCL=1 і

паузи лінія SCL знов повертається в стан 0. На цьому читання байту даних закінчується. Процедури формування стартового, стопового і повторного старту бітів зводяться до встановлення початкових станів сигналів SDA=1, SCL=1 і наступних змін згідно вказаних вище часових діаграм. Використовувані затримки часу (паузи) необхідні для забезпечення надійної передачі. Їх встановлюють, згідно рекомендацій, на період і тривалість сигналу SCL; часу утримання неактивного стану ліній інтерфейсу; на час перед повторним стартом.

6. Програмна реалізація інтерфейсу TWI

Для програмної реалізації інтерфейсу TWI використовується МК ATmega8535 і програмований паралельний порт (ППП) PCA9554 фірми Philips, рис. 9.

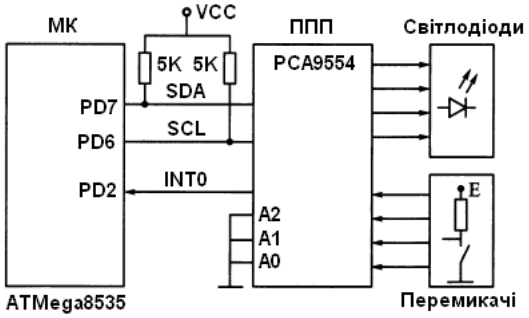
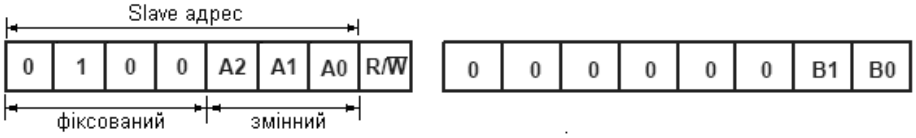


Рисунок 9 – Схема зв’язку МК з ППП по інтерфейсу TWI

ППП є мікросхемою, яка має канал послідовного зв’язку TWI, з однієї сторони, і 8-розрядний паралельний канал введення/виведення, з іншої сторони. Для звернення до ППП в МК системі використовується одна з восьми адрес в діапазоні $20-$27. При цьому три молодших розряди визначають шляхом встановлення сигналів лог. 0 і лог. 1 на входах A2-A0, рис.9, а. Останній розряд R/W визначає операцію: 1 – читання, 0 – запису.



а *б*

Рисунок 10 – Регістр керування (конфігурації) ППП: адресний (а) і командний (б) байти

Після успішного розпізнання адресу посилається командний байт, який записується в керуючий (конфігураційний) регістр ППП, рис. 9, б. Розряди B1, B0 встановлюють операцію (читання/запис) і внутрішній регістр (вхідний, вихідний, зміни полярності або конфігураційний): 0x00 – читання байту з вхідного регістру, 0x01 – читання/запис байту у вихідний регістр, 0x02 – зміна полярності сигналів при читанні/запису байту, 0x03 – конфігураційний регістр (встановлює напрямок передачі розрядів портів I/O, лог. 1 – на вхід, лог. 0 – на вихід). Звернення до порту містить три посилання: адрес, команди, дані. Обмін з портом виконується по запиту переривання, яке формується мікросхемою ППП при зміні

сигналів на входах порту. Для цього вихід INT ППП підключений до входу INT0 МК (для ATmega8535 лінія порту PD2).

В алгоритмі основної програми передбачена наступна послідовність дій:

- ініціалізація порту МК з лініями інтерфейсу;
- налаштування системи переривань МК;
- часова затримка;
- налаштування конфігурації ППП;
- виклик процедури обміну з ППП;
- переведення МК в режим пониженого енергоспоживання і очікування переривань від ППП.

При надходженні запиту виконуються наступні дії:

- ініціалізація обміну з ППП;
- формування стану START і посилання адресу;
- запис у ППП команди введення ($00);
- зміна напрямку обміну;
- повторний старт;
- читання даних з ППП;
- обмін тетрадами;
- зберігання даних;
- формування стану STOP;
- формування стану START і посилання адресу;
- запис у ППП команди виведення ($01);
- зворотне пересилання даних у ППП;
- формування стану STOP і вихід з переривання.

Програмна реалізація інтерфейсу TWI показана в прикладі 1. Програма виконується в середовищі Proteus, рис. 11.

Приклад 1.

```
;*********************************************************************
; 1.asm - робота з програмованим паралельним
;         портом по інтерфейсу TWI.
;4 виводи порту використовуються для введення, 4 - для виведення.
;При зміні сигналів на входах порту на лінії INT0 виробляється
;запис переривання низького рівня.
;За перериванням записуються вхідні значення порту ППП
;і передаються на лінії виходу.
;*********************************************************************
.org 0x000
     rjmp RESET         ;обробка скидання
.org 0x001
     rjmp UPDATE        ;перехід на обробник переривань від ППП

.include "m8535def.inc" ;файл визначень МК
.equ SCLP = 6      ; SCL-pin (порту PD)
.equ SDAP = 7      ; SDA-pin (порту PD)

.def twidelay = r16     ;Лічильник циклу затримки
.def twiadr = r17       ;регістр адресу шини TWI
.def twidata = r18      ;регістр даних  шини TWI
```

```
.equ twird = 1     ;1-читання
.equ twiwr = 0     ;0-запис
;*********************************************************************
RESET:
main:
     ldi r16, HIGH(RAMEND)   ;ініціалізація вказівника стеку
     out sph, r16
     ldi r16, LOW(RAMEND)
     out spl, r16

; ініціалізація інтерфейсу TWI
     clr twidata        ; чистка регістра даних
     out DDRD, twidata
     ldi twidata, 0b0000_0100  ; x04 - порт на вихід
     out PORTD, twidata
; налаштування МК
     cli
     ldi r16, 1<<INT0   ;дозвіл переривання INT0(по спаду)
     out GICR, r16
 ldi r16, (1<<ISC01)|(0<<ISC00)
     out MCUCR, r16
     sei

     rcall twi_hp_delay ; пауза
     rcall twi_hp_delay ; для готовності всіх пристроїв,

; налаштування конфігурації мікросхеми ППП
     ldi twiadr,$40+twiwr    ;посилання адресу ППП+запис
     rcall twi_start         ;генерація стартового біту
     ldi twidata,$03
     rcall twi_write         ;команда запису в порт конфігурації
     ldi twidata,$f0         ;старші  4 біти - на ввід,
     rcall twi_write         ;молодші 4 біти - на вивід
     rcall twi_stop     ;генерація стопового біту

     rcall UPDATE       ;ввід-вивід через ППП
loop:
     sleep              ;перехід в режим пониженого енергоспоживання
     nop
     rjmp loop

;****************************************
; підпрограма обробки переривання від ППП
;****************************************
UPDATE:
     ldi twiadr, $40+twiwr   ;посилання адресу ППП+запис
     rcall twi_start
     ldi twidata, $00
     rcall twi_write         ;команда читання даних з порту
     ldi twiadr, $40+twird   ;зміна напрямку обміну
     rcall twi_rep_start
     set                ;підтвердження після читання не буде
     rcall twi_read     ;читання даних
     swap twidata       ;обмін тетрадами
     mov r20, twidata        ;зберігання
```

```
     rcall twi_stop     ;генерація стопового біту
     clt                ;скидання прапора T
     ldi twiadr, $40+twiwr   ;зміна напрямку обміну
     rcall twi_start
     ldi twidata, $01
     rcall twi_write         ;команда виведення даних у порт
     mov twidata, r20
     rcall twi_write         ;виведення даних
     rcall twi_stop     ;генерація стопового біту
     reti

;*****************************************************************
;           ІМПОРТОВАНІ ПІДПРОГРАМИ
; ЯКІ ВИКОРИСТОВУЮТЬСЯ ДЛЯ РОБОТИ ВЕДУЧОГО МК У ІНТЕРФЕЙСІ TWI.
;*****************************************************************
; Для комунікації використовуються лінії порту PD - PD6(SCL) і PD7(SDA).
; Керування виводами SDA, SCL з відкритим стоком здійснюється
; шляхом попереднього встановлення бітів PORTx в 0
; і в подальшому за допомогою встановлення/скидання бітів напрямку DDRx.
;**** Основні функції: ****
; twi_start - стартове посилання, посилання адресу і напрямку,
; twi_rep_start - посилання "повторного старту" (repeated start),
; twi_write - передача байту,
; twi_read - прийом байту,
; twi_stop – стопове посилання.
;*******************************
; Функції затримок
;*******************************
twi_hp_delay:           ;затримка (на 5мкс мінімум)
     ldi twidelay, 2
twi_hp_delay_loop:
     dec  twidelay
     brne twi_hp_delay_loop
     ret
twi_qp_delay:           ;затримка (на 2.5 мкс мінімум)
     ldi twidelay, 1
twi_qp_delay_loop:
     dec twidelay
     brne twi_qp_delay_loop
     ret

;*********************************************************
; Функція повторного старту готовить шину twi до формування
; стартового біту.
; За даною функцією має слідувати виклик twi_start.
;*********************************************************
twi_rep_start:
     rcall twi_qp_delay
     sbi DDRD, SCLP     ;захоплення лінії SCL (вихід SCL=0)
     cbi DDRD, SDAP     ;звільнення лінії SDA
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL (вихід SCl=1)
     rcall twi_qp_delay ;затримка
;************************************************
; Функція  формування стартового біту.
```

```
; За даною функцією має слідувати виклик twi_write.
;*************************************************
twi_start:
     mov twidata, twiadr
     sbi DDRD, SDAP     ;захоплення SDA
     rcall twi_qp_delay ;затримка
;*************************************************
; Функція запису одного байту у ведений пристрій.
; Також використовується для передачі адресу.
; За даною функцією має слідувати виклик twi_get_ack.
;*************************************************
twi_write:
     sec           ;встановлення прапора C
     rol twidata        ;зсув першого біту в C
     rjmp twi_write_first
twi_write_bit:
     lsl twidata        ;посилання наступного біту в C
twi_write_first:
     breq twi_get_ack   ;перехід, якщо передача закінчена
                   ;(регістр порожній)
     sbi DDRD, SCLP     ;захоплення лінії SCL
     brcs twi_write_high     ;якщо біт встановлений, звільнити SDA
twi_write_low:     ; інакше
     sbi DDRD, SDAP     ; захопити SDA
     rjmp twi_write_delay
twi_write_high:
     cbi DDRD, SDAP     ;звільнення лінії SDA
twi_write_delay:
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL
     rcall twi_hp_delay ;затримка
twi_write_check_wait:
     sbis PIND, SCLP         ;очікування стану SCL=1
     rjmp twi_write_check_wait
     rjmp twi_write_bit

;*********************************************************************
; Функція читання підтвердження. Використовується функцією twi_write.
;*********************************************************************
twi_get_ack:
     sbi DDRD, SCLP     ;захоплення лінії SCL
     cbi DDRD, SDAP     ;звільнення лінії SDA
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL
twi_get_ack_wait:
     sbis PIND, SCLP         ;очікування високого рівня SCL
     rjmp twi_get_ack_wait
     clc                ;скидання прапора C
     sbic PIND, SDAP         ;якщо SDA=1,
     sec                ;встановлення прапора C
     rcall twi_hp_delay ;затримка
     sbi DDRD, SCLP     ;захоплення лінії SCL
     ret
;************************************************************
; Функція  читання одного байту від веденого в регістр twi_data.
```

```
; Значення прапора C=1 використовується як признак кінця прийому.
; Після даної функції має слідувати функція twi_put_ack.
;*****************************************************************
twi_read:
;завантажуємо $01 – це після прийому даних приведе до встановлення прапора C
     ldi twidata, 0x01
twi_read_bit:
     sbi DDRD, SCLP     ;захоплення лінії SCL
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL
twi_read_check_wait:
     sbis PIND, SCLP         ;очікування стану SCL=1
     rjmp twi_read_check_wait
     rcall twi_hp_delay ;затримка
     clc                ;скидання прапора C
     sbic PIND, SDAP         ;якщо SDA=1,
     sec                ;встановлення прапора C
     rol twidata             ;зберігання прийнятого біту
     brcc twi_read_bit       ;приймання закінчено, коли прапор C=1
;*****************************************************************
; Функція формування підтвердження. Використовується функцією twi_read.
; При значенні прапора T=1 підтвердження не формується,
; після чого звичайно формується стоповий біт.
; При T=0 формується підтвердження прийому.
;*****************************************************************
twi_put_ack:
     sbi DDRD, SCLP     ;захоплення лінії SCL
     brtc twi_put_ack_low    ;якщо T=0, формується підтвердження
     cbi DDRD, SDAP     ;звільнення лінії SDA
     rjmp twi_put_ack_high
twi_put_ack_low:
     sbi DDRD, SDAP     ;захоплення SDA
twi_put_ack_high:
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL
twi_put_ack_wait:
     sbis PIND, SCLP         ;очікування звільнення SCL
     rjmp twi_put_ack_wait
     rcall twi_hp_delay ;затримка
     sbi DDRD, SCLP
     ret
;*********************************
; Функція формування стопового біту
;*********************************
twi_stop:
     sbi DDRD, SDAP     ;захоплення SDA
     rcall twi_hp_delay ;затримка
     cbi DDRD, SCLP     ;звільнення лінії SCL
     rcall twi_qp_delay ;затримка
     cbi DDRD, SDAP     ;звільнення лінії SDA
     rcall twi_hp_delay ;затримка
     ret
;*********************************
```

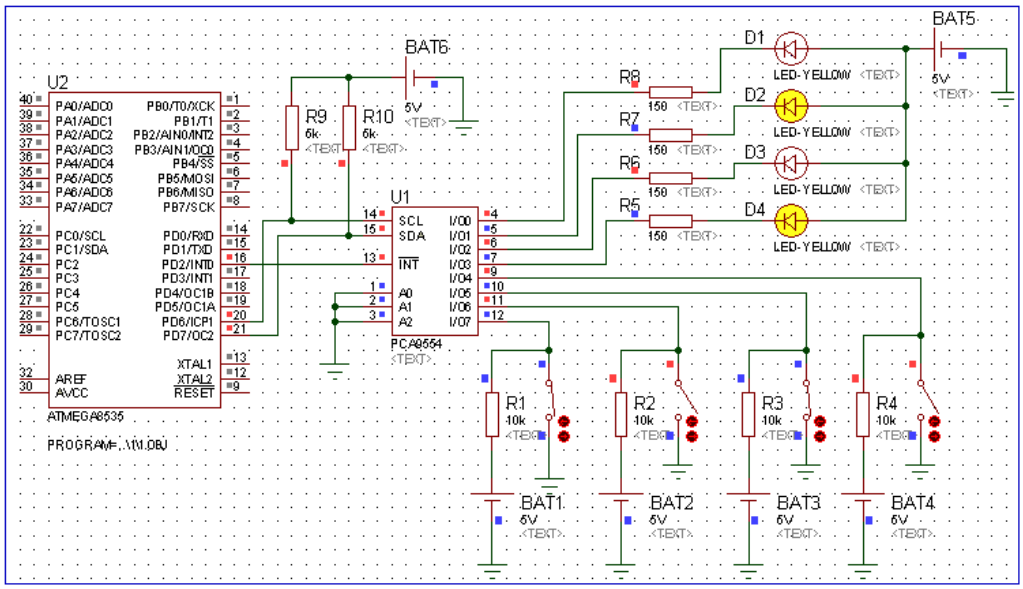


Рисунок 11 – Електрична схема з’єднань для роботи МК з інтерфейсом TWI

7. Приймання/передача даних з використанням апаратного TWI

Процедура передачі одного байту даних від ведучого до веденого пристрою:

1. Перший крок в передачі даних по TWI – встановлення умови START. Для цього в регістрі TWCR встановлюються в лог. 1 біти – переривання TWINT (очищується записом лог.1, TWINT=1), TWSTA (TWSTA=1) і активізації модуля TWI (TWEN=1). Після формування стану START прапор TWINT встановлюється в лог. 1. Для переходу до наступної дії виконується перевірка коду стану $08 в регістрі TWSR ($08 – SLA+W has been transmitted; ACK has been received, SLA+W has been transmitted; NOT ACK has been received). Очікування встановлення прапора TWINT можна замінити обробкою запиту переривання. Функція встановлення умови старту реалізована в процедурі SendStart.

2. Вміст пакету з адресою і нульовим значенням біту напрямку записується в регістр TWDR. Передача ініціалізується скиданням прапора TWINT. Після закінчення передачі адресу і отримання біту підтвердження прапора TWINT встановлюється в 1, а в регістре TWSR встановлюється код статусу $18, ($18 – SLA+W has been transmitted; ACK has been received. $20 – SLA+W has been transmitted; NOT ACK has been received). Функції запису реалізована в процедурі SendAdr.

3. Після перевірки кода статуса в регістр даних TWDR завантажується байт даних для передачі і скидується прапор TWINT. По закінченню передачі даних і отримання біта підтвердження прапор TWINT встанавлюється в лог. 1, а в регістрі TWSR встановлюється код статусу $28 ($28 – Data byte has been transmitted; ACK has been received. $30 – Data byte has been transmitted; NOT ACK has been received). Ці дії забезпечують передачу від ведучого пристрою до веденого як команд, так і даних. Програмна реалізація функції виводу даних (а також команд) реалізована в процедурі SendComData.

4. Після успішного завершення передачі виконуються команди для формування стану Stop. Функція завершення реалізована в процедурі Stop.

Процедура прийому одного байту даних ведучим пристроєм:

1. В регістр TWCR виводиться команда для формування стану Start; після формування стану Start прапор TWINT встановлюється в лог. 1. Для переходу до наступної дії перевіряється код стану ($08) в регістрі TWSR. (Очікування встановлення прапора TWINT також можна замінити обробкою запиту переривання.);

2. Вміст пакету з адресою і одиничним значенням біту напрямку записується в регістр TWDR і ініціалізується передача. Після закінчення передачі адресу і отримання біта підтвердження прапор TWINT встановлюється в 1, а в регістрі TWSR встановлюється код статусу ($40);

3. Після скидання прапора TWINT (дозвіл прийому) і отримання байту даних від веденого дані з регістра TWDR переписуються в один з регістрів загального призначення. При успішному прийомі код статусу в регістрі TWSR приймає значення $50. При необхідності формується біт підтвердження прийому;

4. Після закінчення прийому виконується команда для формування стану Stop.

Процедура прийому одного байту даних веденим пристроєм:

1. В регістр TWCR виводиться команда, скидається прапор TWINT;

2. Після прийому першого байта з адресою веденого пристрою формується запит переривання, перевіряється код статусу в регістрі TWSR; якщо він дорівнює $60, прийом власного адреса виконано успішно;

3. Передається біт підтвердження шляхом встановлення TWEN=1 і скидається прапор TWINT;

4. Перевіряється код статусу в регістрі TWSR (якщо він дорівнює $80, значить в регістр TWDR прийнято байт даних); передається біт підтвердження (TWEN=1) і скидається прапор TWINT;

Дії 4-го кроку повторюються, пока не будуть прийняті всі повідомлення. При поступленні стану Stop в регістрі TWSR формується код статусу $A0 (кінець пакету).

Аналогічно можна подати дії веденого пристрою при передачі даних. Детальний опис Всі варіанти обміну і значення статусних кодів детально описані в технічній документації МК.

8. Обмін даними між МК і паралельним портом PCA9554

Програмна реалізація обміну даними між МК ATmega8535 з вбудованим модулем TWI і паралельним портом PCA9554 показана в прикладі 2. Програма виводить в порт інвертований 8-розрядний код. При передачі пакетів з адресами і даними використовується механізм програмної перевірки умов встановлення в лог. 1 прапора TWINT і станів регістра TWSR. Програма виконується в середовищі Proteus, рис. 12.

Приклад 2.

```
;*****************************************************************
; 2.asm - вивід даних у
;програмований паралельний порт по інтерфейсу TWI
;*****************************************************************
.org 0x000
     rjmp init
```

```
;.include "m8535def.inc"         ;файл визначень ATmega8535

.equ twird = 0x01           ;1-читання
.equ twiwr = 0x00           ;0-запис
;.equ SLA_W = 0b00100_0000  ;адресний байт (slave addr + write)
.equ SLA_W =0x44+twiwr         ; 0x40, 0x42, 0x44, …
.equ TWBRATE = 12 ; Для частоти Fclk = 4 МГц TWBR=12 TWPS=0: Fscl=100 КГц
;**** коди статусу в режимі ведучий передавач ****
.equ START       = 0x08 ;після старту
.equ MT_SLA_ACK  = 0x18 ;після передачі адресу і підтвердження
.equ MT_DATA_ACK = 0x28 ;після передачі даних  і підтвердження
.def DATA = r18
init:
     ldi r16, HIGH(RAMEND)   ;ініціалізація вказівника стеку
     out sph, r16
     ldi r16, LOW(RAMEND)
     out spl, r16
;підготовка TWI до роботи
     ldi r16,TWBRATE
     out TWBR, r16
     ldi r22,0xAA     ; дані 0b1010_1010
; налаштування мікросхеми ППП
     rcall Send_Start   ;генерація стартового біта
     ldi r16, SLA_W     ;посилання адресу+запису
     rcall Send_Adr    ; адрес 0x20 на запис
     ldi DATA, 0x03;командний файл - вибір конф. регістра
     rcall Send_Data
     ldi DATA, 0x00; запис у конф. регістр 0x00 - порти I/O на вихід
     rcall Send_Data   ;                        0x01 - порти I/O на вхід
     rcall Stop         ;генерація стопового біту

loop:
; виведення даних у ППП
     rcall Send_Start   ;генерація повторного стартового біту
     ldi r16, SLA_W     ;посилання адресу+запису
     rcall Send_Adr
     ldi DATA,$1        ;командний файл - вибір рег. вихідних портів
     rcall Send_Data
     mov DATA,r22  ; байт даних
     rcall Send_Data
     rcall Stop         ;генерація стопового біту
    com r22        ;інвертування  0b0101_0101
; пауза
    rcall DELAY
     rjmp loop     ;повторення виводу
;*****************************************************************
; Функції режиму ведучого передавача
;*****************************************************************
Send_Start:
ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
out TWCR, r16      ;умова для стану START
;очікування встановлення прапора TWINT
wait1:
in r16, TWCR
sbrs r16, TWINT
```

```
rjmp wait1
;перевірка виконання старту
in   r16, TWSR
andi r16, 0xF8
cpi  r16, START    ; 0x08
brne error
ret

Send_Adr:
out TWDR, r16      ;завантаження пакету з адресою
ldi r16, (1<<TWINT)|(1<<TWEN)
out TWCR, r16      ;передача
;очікування встановлення прапора TWINT
wait2:
in r16, TWCR
sbrs r16, TWINT
rjmp wait2
;перевірка виконання передачі (отримання ACK)
in r16, TWSR
andi r16, 0xF8  ; 0xF9
cpi r16, MT_SLA_ACK     ; 0x18
brne error
ret

Send_Data:
out TWDR, Data     ;посилання командного байту
ldi r16, (1<<TWINT)|(1<<TWEN)
out TWCR, r16      ;передача
wait3:
in r16, TWCR
sbrs r16, TWINT         ;очікування встановлення прапора TWINT
rjmp wait3
;перевірка виконання передачі
in r16, TWSR
andi r16, 0xF8
cpi r16, MT_DATA_ACK     ; 0x28
brne error
ret

Stop:
ldi r16, (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
out TWCR, r16      ; умова для стану STOP
ldi r16,0x1f
wait4:
dec r16
brne wait4
ret

;*** Затримка довга ***
DELAY:   ldi r19,10   ; Величина затримки підбирається для кожного ПК!
d1:  ldi r20,0x3f
d2:  ldi r21,0x5f
d3:  dec r21
         brne d3
         dec r20
```

```
        brne d2
        dec r19
        brne d1
        ret

error: ret
;***********************************************************************
```

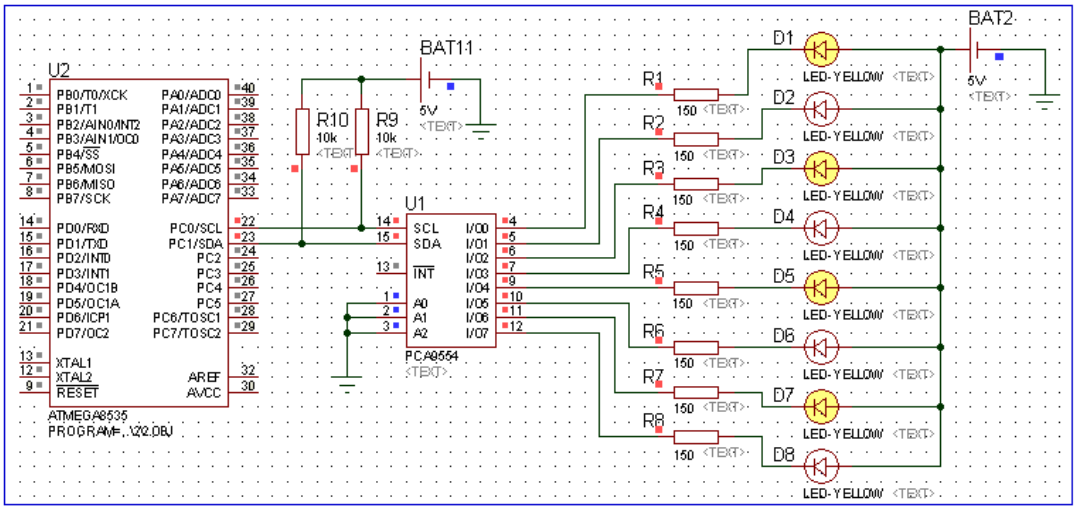


Рисунок 12 – Електрична схема з’єднань для роботи МК з паралельним портом PCA9554

9. Обмін даними між двома МК з використанням інтерфейсу TWI

Передача повідомлень між МК1 і МК2 здійснюється з використанням модуля TWI. МК1 виступає як ведучий, а МК2 як ведений. МК2 працює в режимі очікування. Після передачі повідомлення можна продивитися коди символів, натискаючи кнопку SW0. Програмна реалізація обміну даними між двома МК ATmega8535 показана в прикладі 3. В програмі використовуються функції *Send_Start, Send_Adr, Send_Data, Stop* ті ж, що і в програмі 2.asm. Програми виконуються на двох платах STK500 або в середовищі Proteus, рис. 13. При використанні плат STK500 замість зовнішніх резисторів можна використати внутрішні підтягуючі резистори МК2.

Приклад 3.

```
;**************************************
; 3.asm - передача повідомлення ведучим МК1 по інтерфейсу TWI
;**************************************
.org 0x000
    rjmp init
.include "m8535def.inc" ;файл визначень ATmega8535
.def temp  = r16
.def SLA_W = r17        ;адресний байт
.def DATA  = r18        ;байт даних (команди)
.def count = r19        ;лічильник даних
.equ twiwr = 0          ;0-запис
.equ ADR = 0x44         ; 0x40, 0x42, 0x44, ...
;**** коди статусу в режимі ведучий передавач ****
.equ START       = 0x08 ;після старту
.equ MT_SLA_ACK  = 0x18 ;після передачі адресу
```

```
.equ MT_DATA_ACK = 0x28 ;після передачі даних
init:
    ldi r16, HIGH(RAMEND) ;ініціалізація вказівника стеку
    out sph, r16
    ldi r16, LOW(RAMEND)
    out spl, r16
     ldi ZL,low(text*2) ;завантаження адресу тексту
     ldi ZH,high(text*2)     ; повідомлення в регістр Z
     ldi count,3             ;встановлення лічильника передач
;підготовка TWI до роботи
    ldi r16,12           ;для частоти Fclk = 4 МГц TWBR=12 TWPS=0
    out TWBR, r16        ; Fscl=100 КГц
OUTPUT:                     ;вивід даних
    rcall Send_Start        ;генерація стартового біту
    ldi SLA_W, ADR+twiwr    ;посилання адресу + запис
    rcall Send_Adr
     lpm          ;зчитування байту з flash-пам'яті в r0
     mov DATA, r0   ;вивід даних
     rcall Send_Data
    rcall Stop           ;генерація стопового біту
; затримка
   ldi r21,0xff
dd: dec r21
    brne dd;
    adiw zl,1  ;збільшення адресу на 1
     dec count   ;зменшення лічильника на 1
     brne OUTPUT
loop: rjmp loop
text: .db 'A','V','R' ;текст повідомлення (коди $41,$56,$52)
;*********************************************************************
; Функції режиму ведучого передавача (аналогічні до 2.asm)
;*********************************************************************
Send_Start:
    ret
Send_Adr:
    ret
Send_Data:
    ret
error:
    ret

;*********************************************************************
; 4.asm - приймання по інтерфейсу TWI для веденого МК2
;*********************************************************************
.org 0x000
     rjmp init
.org 0x011
     rjmp TWI_INT   ;перехід до обробки переривання
.include "m8535def.inc"  ;файл визначень МК ATmega8535
.def count = r22  ;лічильник даних
.def temp = r16
.equ SHOW=0
.equ ADR = 0x44         ; 0x40, 0x42, 0x44, ...
;****коди статусу в режимі веденого***
.equ TW_SR_SLA_ACK = 0x60   ;прийнятий байт з адресою
```

```
.equ TW_SR_DATA_ACK = 0x80  ;прийнятий байт даних
.equ TW_SR_STOP = 0xA0      ;виявлено стан STOP
;***Ініціалізація веденого МК***
init:
     ldi r16, HIGH(RAMEND)
     out sph, r16
     ldi r16, LOW(RAMEND)
     out spl, r16
     clr temp      ;налаштування
     out DDRB,temp ; на введення
     sbi PORTB,0        ; в порт PB0
     ser temp      ;налаштування
     out DDRD,temp ; на виведення
     out PORTD,temp; виводів порту PD
     ldi count,0x03;встановлення лічильника байтів
     ldi XL,0x80        ;в регістрі X адрес, за яким
     ldi XH,0x01        ; записуються прийняті дані
;підготовка модуля TWI до роботи
     clr temp
     out DDRC,temp     ; на ввід
     ldi temp,0x03 ;підключення підтягуючих резисторів
     out PORTC, temp    ; на лінії SCL,SDA
     ldi r16,12         ;для Fclk = 4 МГц TWBR=12 TWPS=0
     out TWBR, r16 ; Fscl=100КГц
     ldi r16,ADR   ;адрес пристрою TWI (довільний)
     out TWAR,   r16
     ldi r16, (1<<TWINT)|(1<<TWEA)|(1<<TWIE)|(1<<TWEN)
     out TWCR, r16      ;Запуск обміну по TWI
     sei
loop:    rjmp loop     ;нескінченний цикл очікування
;*****************************************************************
; обробник переривання від модуля TWI
;*****************************************************************
TWI_INT:
     in r16, TWSR
     cpi r16, TW_SR_SLA_ACK  ;1-а перевірка  - 0x60
     brne ierror
     ldi r16, (1<<TWINT)|(1<<TWEA)|(1<<TWEN)
     out TWCR, r16
TW_WAIT_DATA:
     in r16, TWSR
     cpi r16, TW_SR_DATA_ACK ;2-а перевірка  - 0x80
     brne TW_WAIT_DATA
     in r16, TWDR
     st X+,r16          ;зберігання байту даних у пам'яті
     ldi r16, (1<<TWINT)|(1<<TWEA)|(1<<TWEN)
     out TWCR, r16
TW_WAIT_STOP:      ;очікування стану STOP
     in r16, TWSR
     cpi r16, TW_SR_STOP     ;3-я перевірка - 0xA0
     brne TW_WAIT_STOP
     ;байт даних прийнятий
     dec count      ; зменшення лічильника даних
     brne ierror        ; якщо не нуль, продовження,
     rcall OUTLED  ; інакше виведення на індикатори
```

```
     reti
ierror:
     ldi r16, (1<<TWINT)|(1<<TWEA)|(1<<TWIE)|(1<<TWEN)
     out TWCR, r16 ;перезапуск інтерфейсу
     reti
; **********
OUTLED:
     clr temp       ; сигналізація -
     out PORTC,temp; прийом закінчений
     ldi XL,0x80
     ldi count,3        ;встановлення лічильника байтів
WAIT_SHOW: sbic PINB,SHOW    ;очікування натискання
     rjmp WAIT_SHOW; кнопки SHOW
     ld temp, X+        ;зчитування байту з пам'яті
     com temp       ;інвертування і
     out PORTD,temp;виведення на світлодіоди
     rcall DELAY        ;затримка
     dec count      ;якщо показані не всі дані,
     brne WAIT_SHOW; то продовження по натисканню SHOW
     ret
;***Затримка***
DELAY:    ldi r19,10
     ldi r20,255  ; 60
     ldi r21,255  ; 10
dd: dec r21
     brne dd
     dec r20
     brne dd
     dec r19
     brne dd
     ret
```

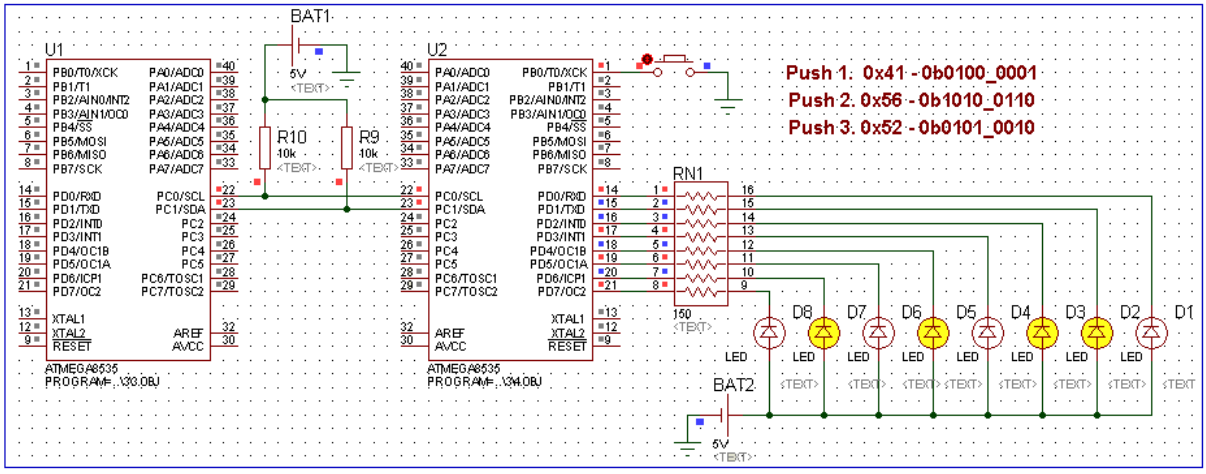


Рисунок 13 – Електрична схема з’єднань для передачі повідомлень від МК1 до МК2

Завдання.

1. Дослідити роботу інтерфейсу TWI використовуючи приклади 1, 2, 3.

2. Написати і налагодити програму для режиму ведучого МК з прийомом даних по інтерфейсу TWI, використовуючи вбудований модуль TWI.

3. Написати і налагодити програму для ведучого МК, який вводить 4-розрядний код через молодші розряди порту ППП і виводить прийняті дані через старші розряди порту ППП по інтерфейсу TWI, використовуючи вбудований модуль TWI.

Питання.

1. Для чого призначений двопровідний послідовний інтерфейс TWI?
2. Послідовність передачі даних і формати пакетів в інтерфейсі TWI.
3. Алгоритми читання і запису байту в інтерфейсі TWI.
4. Як змінити програму виводу в програмований паралельний порт, щоб дозволити переривання від модуля TWI?
5. Як працює паралельний порт PCA9554 і які він містить регістри?

ЛАБОРАТОРНА РОБОТА № 12. Аналоговий компаратор

Мета роботи: практичне вивчення оброблення аналогових сигналів за допомогою аналогового компаратора.

Обладнання: Microchip Studio, навчальна плата STK500, МК Atmega.

1. Теоретична частина

1.1. Аналоговий компаратор

Компаратор – це пристрій, який порівнює дві напруги. В залежності від форми порівнюваних сигналів компаратори поділяються на аналогові і цифрові.

Аналогові компаратори порівнюють напругу – аналогову величину. Аналоговий компаратор, по суті, є операційним підсилювачем з нескінченним підсиленням. Аналоговий компаратор має два входи і один вихід. Вихід перемикається на VDD або на землю (0 В) залежно від відносного рівня напруги на двох вхідних контактах. Як і в операційному підсилювачі, два входи позначаються як інвертуючий (-) вхід (INV) і неінвертуючий (+) вхід (NINV). Коли неінвертуюча вхідна напруга вища за напругу інвертуючого входу, вихід перемикається на VDD, а якщо нижча, то вихід перемикається на землю. Таким чином отримуються два логічні стани, рис. 1

Vout = 1, якщо V+ > V-
Vout = 0, якщо V+ < V-

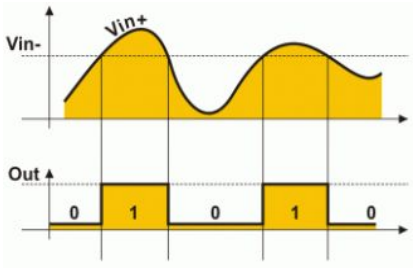
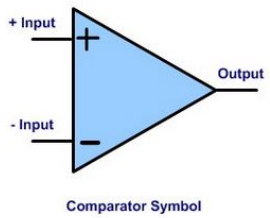
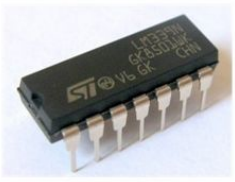


Рисунок 1 – Вхідні і вихідні сигнали компаратора

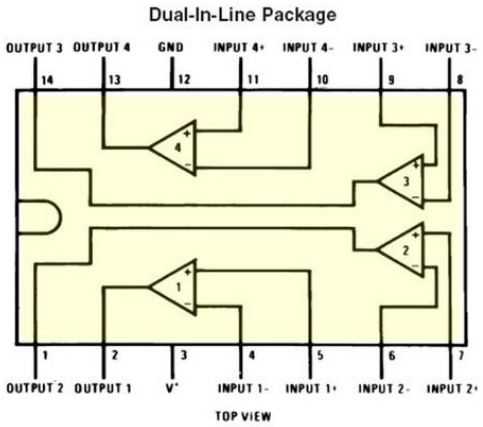
Аналогові компаратори використовуються як окремі мікросхеми (рис. 2) або як вбудовані блоки мікроконтролерів (МК) , рис. 3.



Comparator Symbol



LM339 Quad IC



LM339 Quad Compartor

Рисунок 2 – Чотири незалежних компаратори в складі одної IC LM339

Найпростіше взаємодіяти з аналоговим сигналом з використанням аналогового компаратора вбудованого в МК. Блок схема аналогового компаратора МК ATMega показана на рис. 3.

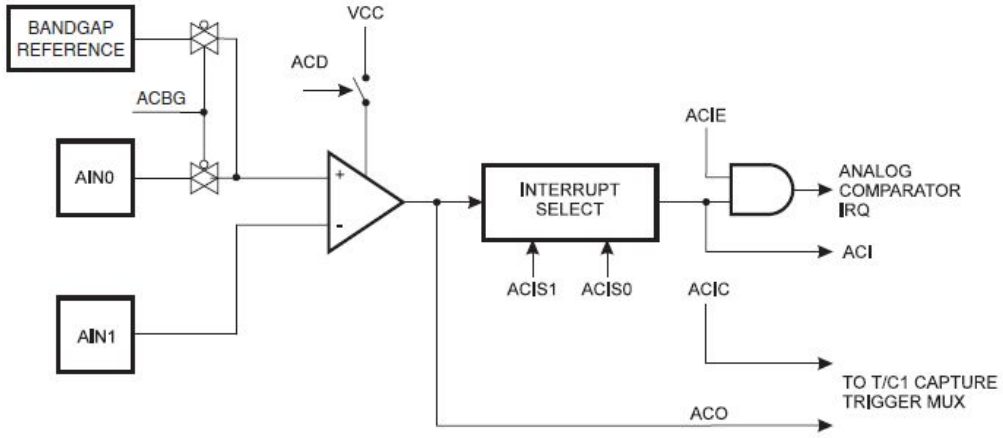


Рисунок 3 – Блок схема аналогового компаратора вбудованого в МК

Аналоговий компаратор порівнює вхідні значення на позитивному вході AIN0(+) і негативному вході AIN1(-). Можна вибрати будь-який з контактів АЦП [7..0], щоб замінити негативний вхід аналогового компаратора. Якщо напруга на контакті AIN0 більша від напруги на контакті AIN1, вихід аналогового компаратора ACO встановлюється в 1. Вихід компаратора може запускати функцію захоплення Input Capture лічильника Timer/Counter1. Додатково компаратор може збуджувати своє переривання. Користувач може налаштувати збудження переривання виходу компаратора за *наростаючим* або *спадаючим* фронтом сигналу, або при перемиканні *рівня* сигналу.

Таким чином аналоговий компаратор дозволяє не тільки фіксувати перевищення вхідного значення відносно порогового, але і момент часу коли це трапляється.

Для керування роботою і відображення стану аналогового компаратора використовується регістр **ACSR** (Analog comparator and status register.)

7	6	5	4	3	2	1	0
ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

• ACD (Analog Comparator Disable): аналоговий компаратор при ACD=0 ввімкнено, при ACD=1 вимкнено. При зміні ACD біта необхідно очистити біт ACIE, щоб запобігти виникненню переривання.

• ACBG (Analog Comparator Bandgap Select): при ACBG=0 до AIN0(+) підключається зовнішній сигнал, ACBG=1 — підключається внутрішня опорна напруга 1,26 В.

• ACO (Analog Comparator Output): результат порівняння на синхронізованому виході підключеного до ACO. Синхронізація вносить затримку на 1-2 такти.

• ACI (Analog Comparator Interrupt Flag): прапор переривань від компаратора встановлюється апаратно, залежно від значень бітів ACIS1, ACIS0.

• ACIE (Analog Comparator Interrupt Enable): ACIE=1 і SREG.I=1 дозвіл переривань компаратора, інакше – заборона.

• ACIC (Analog Comparator Input Capture): підключення компаратора до схеми захоплення таймера лічильника 1 (1- дозвіл, 0 - заборона).

• ACIS1, ACIS0 (Analog Comparator Interrupt Mode Select): умови виникнення переривань.

ACIS1	ACIS0	Умови виникнення переривань
0	0	Будь-яка зміна стану виходу компаратора
0	1	Резерв
1	0	Переривання по спадаючому фронту виходу компаратора (з одиниці на нуль)
1	1	Переривання по наростаючому фронту виходу компаратора (з нуля на одиницю)

Входи компаратори під'єднані до виводів PB2, PB3, тому відповідні порти мають бути налаштовані як входи з підтягуючими резисторами.

Приклад:

```
ldi r16,0b1000_1000
out ACSR,r16
```

В регістрі ACSR біти ACD і ACIE встановлені в 1. В результаті компаратора буде відключений, але переривання залишаться активними. При включенні компаратора можуть генеруватися фальшиві переривання. Тому *рекомендується* забороняти переривання при включенні/відключенні компаратора за допомогою біту ACD.

На інвертуючий вхід компаратора може надходити сигнал з виходу мультиплексора модуля АЦП. Для цього потрібно встановити в одиницю розряд ACME, який знаходиться, в залежності від моделі, або в регістрі спеціальних функцій SFIOR, або в регістрі керування АЦП ADCSRB. Модуль АЦП при цьому має бути вимкнено (розряд ADEN регістра ADCSRA треба скинути в нуль).

Модель	Регістр
ATmega8515x/8535x	SFIOR
ATmega8x/16x/32x/64x/128x	
ATmega8x/16x/32x/64x/128x	
ATmega48x/88x/168x	
ATmega162x/164x/165x	ADCSRB
ATmega325x/3250x/645x/6450x	

Приклад 1. Компаратор з порогом 1,23 В, рис. 4.

```
;====================================
; Comparator with 1.23 V threshold
;====================================
 .EQU AINpin = PA3
 .EQU LED    = PD0

.CSEG
.org 0x00
rjmp RESET
```

```
      .org 0x020
      rjmp ANA_COM
      .org 0x28
      reti

INIT:
      cbi DDRA,AINpin
      cbi PORTA,AINpin
      sbi DDRD,LED
      sbi PORTD,LED
      ;-------- redirect AIN1 -> ADC3 ------
      clr r16
      ldi r16,(1<<ACME)
      out SFIOR,r16
      cbi ADCSRA,ADEN
      cbi ADMUX,MUX2
      sbi ADMUX,MUX1
      sbi ADMUX,MUX0
      ;-------------------------------------
      out ACSR,r16
      cbi ACSR,ACD      //Comparator ON
      sbi ACSR,ACBG     //Connect 1.23V reference to AIN0
      sbi ACSR,ACIE     //Comparator Interrupt enable
      cbi ACSR,ACIC     //input capture disabled
      cbi ACSR,ACIS1    //set interrupt on output toggle
      cbi ACSR,ACIS0
      sei               //SREG.I=1
      ret

      // Interrupt handler for ANA_COMP_vect
ANA_COM:
      sbic ACSR,ACO  ; comparator output
      rjmp elseA
      cbi PORTD,LED
      reti
elseA:
      sbi PORTD,LED
      reti

RESET:
      ldi r16,high(RAMEND)
      out SPH,r16
      ldi r16,low(RAMEND)
      out SPL,r16
      call INIT
      main:
            rjmp main    // Infinite loop; interrupts do the rest
```

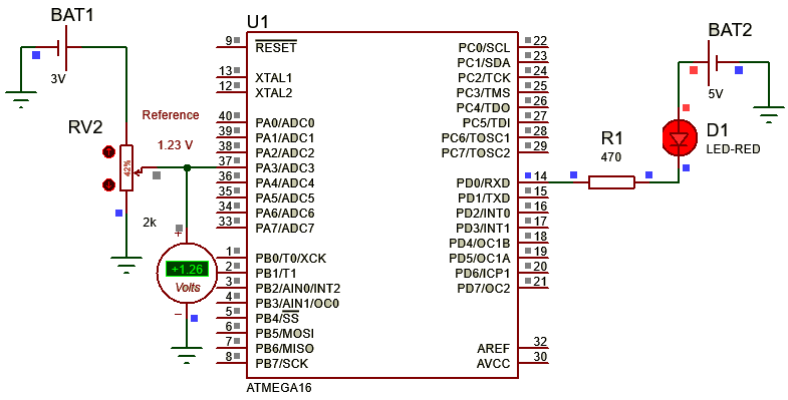


Рисунок 4 — Компаратор з порогом

Приклад 2. Компаратор порівняння напруг, рис. 5

На схемі рис. 4 два вхідних сигнали підключені до контактів AIN0 і AIN1 порту PORTB.

Вихід компаратора відображається світлодіодом, який підключений до контакту PC7.

Коли напруга на контакті AIN0 перевищує напругу на контакті AIN1, вихідний світлодіод буде світитися, інакше світлодіод згасне.

Кроки програмування для прийняття негативного входу AIN1:

1. Дозволити AIN1 для негативного входу, очистивши біт ACME регістра SFIOR.
2. Очистити регістр ACSR.
3. Контролювати біт ACO регістру ACSR і при його встановленні виконати відповідну дію.

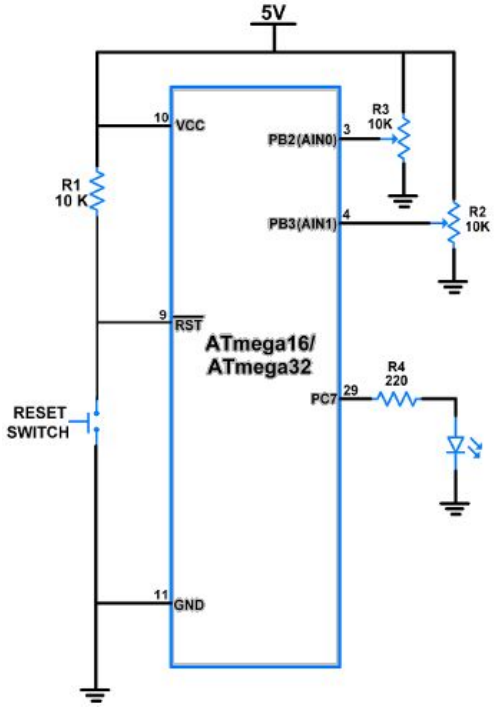


Рисунок 5 – Компаратор порівняння напруг

```
/* Comparator.c */

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

int main()
{
        DDRC |= 0x80;           /* Make pin 7 of PORTC is output */
        SFIOR &= (1<<ACME);     /* Enable AIN1 for -ve input */
        ACSR &= 0x00;           /* Clear ACSR register */

        while(1)
        {
                if (ACSR & (1<<ACO))/* Check ACO bit of ACSR register */
```

```
                PORTC = 0x80;    /* Turn ON PC7 pin */
                else             /* If ACO bit is zero */
                PORTC = 0x00;    /* Then turn OFF PC7 pin */
        }
}
```

Кроки програмування для отримання негативного входу з каналу ADC (АЦП):

1. Вимкнути ADC, очистивши біт ADEN регістра ADCSRA.
2. Вибрати ADC вивід, встановивши регістр ADMUX.
3. Дозволити ADC для негативного входу, встановивши біт ACME регістра SFIOR.
4. Очистити регістр ACSR.
5. Відстежуйте біт ACO регістру ACSR і при його встановленні виконати відповідну дію.

```
/* comparator_using_ADC.c */

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

int main()
{
        DDRC |= 0x80;            /* Make pin 7 of PORTC as output */
        ADCSRA &= (1<<ADEN);     /* Disable ADC */
        ADMUX = 0x00;            /* Select ADC0 as a -ve pin of comparator */
        SFIOR |= (1<<ACME);      /* Enable analog comparator */
        ACSR = 0x00;             /* Clear ACSR resister */

        while(1)
        {
                if (ACSR & (1<<ACO))/* Check ACO bit of ACSR register */
                PORTC = 0x80;    /* Turn ON PC7 pin */
                else             /* If ACO bit is zero */
                PORTC = 0x00;    /* Then turn OFF PC7 pin */
        }
}
```

2. Практична частина

Завдання.

1. Переписати програму прикладу 1 на AVR асемблер.
2. Переписати програму прикладу 2 на AVR асемблер.
3. Використовуючи компаратор виміряти ємність конденсатора, рис. 6.

$v(t)=V_{cc}(1-\exp(-t/T)$ (1)

$T=R_2*C$ (2)

$R_2>>100$ Ом

Алгоритм.

1. Встановити порт PORTnx(AIN-) як вхідний.
2. Сконфігурувати AC і Timer1.
3. Встановити порт PORTnx(AIN+) як вихідний і записати “0” (розрядити конденсатор).

4. Встановити порт PORTnx(AIN+) як вхідний і START Timer1. Конденсатор почне заряджатися.

...

Переривання спрацює коли напруга на конденсаторі V_+ стане рівною V_-:

$v(t)=V_{cc}*R_3/(R_3+R_4)$. (3)

Визначення часу захоплення Timer1

1. Прочитати ICR1 регістр.
2. Перетворити ICR1 в sec => t.
3. Обчислити C з рівнянь (1), (2), (3).

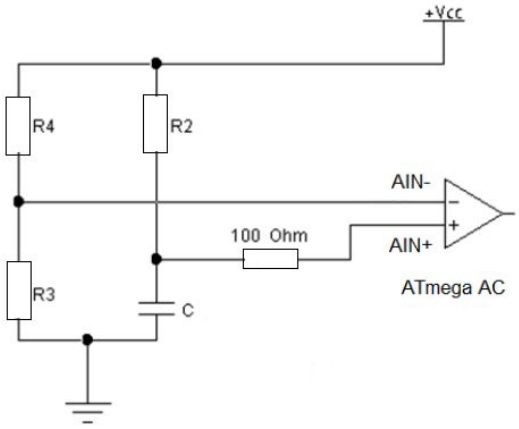


Рисунок 6 – Схема для вимірювання ємності конденсатора

Питання.

1. Принцип роботи аналогового компаратора.
2. Пояснити структуру аналогового компаратора.
3. Пояснити формати керуючих регістрів аналогового компаратора.
4. Як підключити мультиплексор АЦП до компаратора?
5. Де зберігається вихід компаратора?
6. Які умови спричиняють переривання аналогового компаратора?
7. Як підключити вихід компаратора до схеми захоплення таймера лічильника 1?

ЛАБОРАТОРНА РОБОТА № 13. Аналого-цифрові перетворювачі

Мета. Вивчення аналого-цифрових перетворювачів на основі МК.

1. Теоретична частина

1.1. Аналого-цифровий перетворювач

Аналого-цифровий перетворювач (АЦП, Analog to Digit Converter, ADC) дає еквівалентне подання аналогового сигналу у цифровому (двійковому) коді, рис. 1.

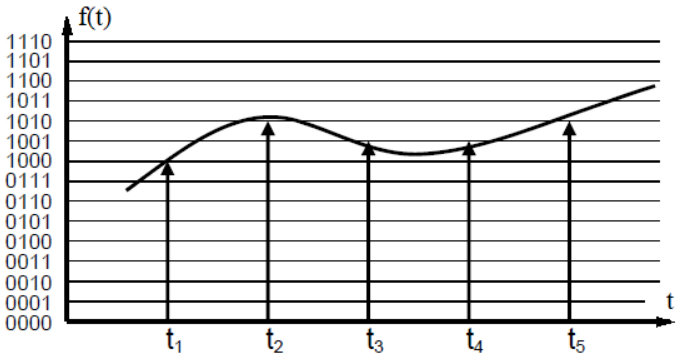


Рисунок 1 – Квантування та дискретизація аналогового сигналу

Основними параметрами будь-якого АЦП є розрядність та максимальна частота дискретизації. Розрядність АЦП визначає кількість рівнів квантування, якими можна подати аналоговий сигнал. 10-ти розрядний АЦП забезпечує 2^{10}=1024 рівнів, тобто діапазон значень на виході перетворювача 0…1023. При розрядності АЦП 10 біт і діапазоні вхідної напруги від 0 до 5 В, розрядність за напругою становить (5-0)/1024≈4,88 мВ. Частота дискретизації визначає частоту вибірки цифрових значень з аналогового сигналу. Розрядність та швидкодія взаємопов'язані параметри: більша розрядність – нижча швидкодія, і навпаки.

Звичайно МК мають 8 аналогових входів. Входи обробляються послідовно один за одним за допомогою мультиплексора. Оскільки реальні АЦП не можуть виконати аналого-цифрове перетворення миттєво, вхідне аналогове значення повинне утримуватися постійним принаймні від початку до кінця процесу перетворення (цей інтервал часу називають *час перетворення*). Для цього на на вході АЦП використовують пристрій вибірки-зберігання. Пристрій зберігає вхідну напругу в конденсаторі, з'єднаному із входом через аналоговий ключ: при замиканні ключа відбувається вибірка вхідного сигналу (конденсатор заряджає до вхідної напруги), при розмиканні – зберігання.

Дійсне перетворення сигналу у цифрову форму виконує схема АЦП послідовного наближення, яка містить допоміжний цифро-аналоговий перетворювач (ЦАП), регістр послідовних наближень, компаратор та блок синхронізації. Блок схема АЦП МК ATMega 8535 показана на рис. 2.

АЦП ATMega 8535 має наступні особливості:

• Розрядність АЦП 10-біт (тому результат перетворення зберігається у двох регістрах ADCH:ADCL).

• Інтегральна нелінійність 0,5 LSB (least significant bit, молодший біт).

• Абсолютна похибка 2 LSB.

• Час перетворення 65-260 мкс.
• Швидкодія до 15 тис. вибірок за секунду.
• 8-мультиплексованих входів.
• 7-диференційних входів.
• 2-диференціальні входи з коефіцієнтом підсилення 10X або 200X.
• АЦП може функціювати коли процесор знаходиться в режимі сну (sleep mode).
• АЦП може генерувати запит на переривання по завершенню перетворення.
• АЦП має внутрішнє джерело опорної напруги 2.26 В.
• Неперервний і одиночний режим перетворення.

1.1. АЦП перетворювачі

Аналого-цифрові перетворювачі (АЦП) є пристроями, які приймають вхідні аналогові сигнали та генерують відповідні їм цифрові сигнали, які можуть оброблятии МК та інші цифрові пристрої.

Багатоканальний АЦП входить в більшість сучасних моделей МК AVR. Звичайно число каналів дорівнює 8, але в різних моделях воно може варіювати від 4 каналів в молодших моделях родини Tiny, 6 каналів в ATmega8 і до 16 каналів в ATmega2560. Багатоканальність означає, що на вході єдиного модуля АЦП встановлений аналоговий мультиплексор, який може підключати цей вхід до різних виводів МК для здійснення вимірювань декількох незалежних аналогових величин з рознесенням по часу. Входи мультиплексора можуть працювати окремо (в несиметричному режимі для виміру напруги відносно "землі") або (в деяких моделях) об'єднуватися в пари для вимірювання диференціальних сигналів. Іноді АЦП додатково забезпечується підсилювачем напруги з фіксованими значеннями коефіцієнта підсилення 10 і 200.

Сам АЦП є перетворювачем послідовного наближення з пристроєм вибірки-зберігання і фіксованим числом тактів перетворення, рівним 13 (або 14 для диференційного входу). Перше перетворення після ввімкнення займає 25 тактів для ініціалізації АЦП. Тактова частота формується аналогічно тому, як це робиться для таймерів – за допомогою спеціального дільника тактової частоти МК, з коефіцієнтом ділення від 1 до 128. Але на відміну від таймерів, вибір тактової частоти АЦП не зовсім довільний, так як швидкодія аналогових компонентів обмежена. Тому коефіцієнт ділення потрібно вибирати таким, щоб при заданій частоті роботи МК тактова частота АЦП вкладалася в рекомендований діапазон 50-200 кГц (тобто максимум близько 15 тис. вимірювань в секунду). Збільшення частоти вибірки допустимо, якщо не потрібно досягнення високої точності перетворення. Роздільна здатність АЦП в МК AVR – 10 двійкових розрядів, чого для більшості типових застосувань досить. Абсолютна похибка перетворення залежить від ряду факторів і в ідеальному випадку не перевищує ±2 молодших розряди, що відповідає загальній точності вимірювання приблизно 8 двійкових розрядів. Для досягнення цього результату необхідно приймати спеціальні заходи: не тільки правильно підбирати тактову частоту в рекомендований діапазон, але і знижувати по максимуму інтенсивність цифрових шумів. Для цього рекомендується, як мінімум, не використовувати невикористані виводи того ж порту, до якого підключений АЦП, робити правильну розводку друкованої плати і додатково включати спеціальний режим пониження шуму (ADC Noise Reduction). АЦП може працювати у двох режимах: окремого перетворення або вільного запуску. Другий режим доцільний лише при максимальній частоті вибірок. В інших випадках його слід уникати, оскільки обійти в цьому

випадку необхідність паралельної обробки цифрових сигналів, як правило, неможливо, а це означає зниження точності перетворення.

Блок схема АЦП показана на рис. 2.

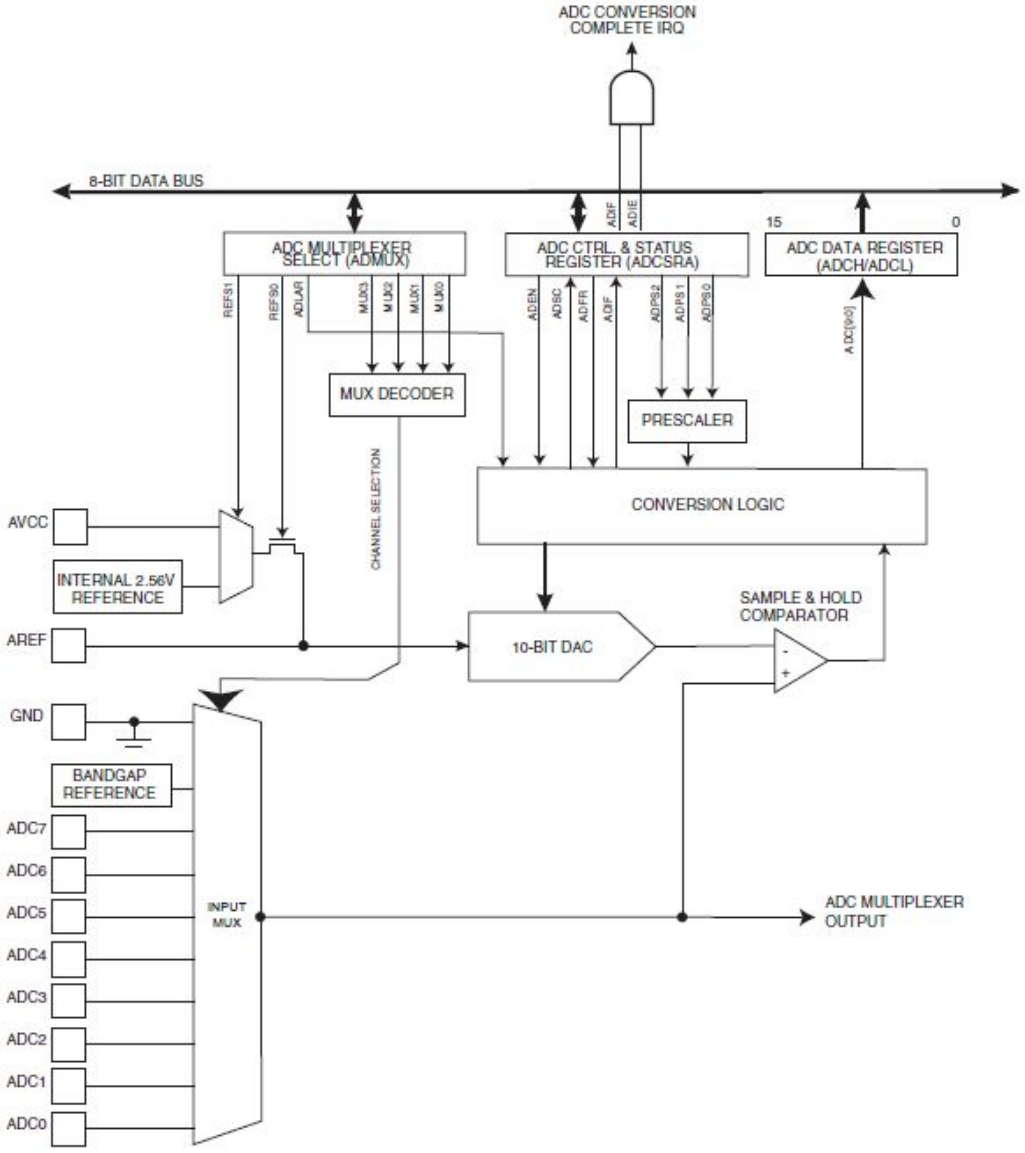


Рисунок 2 – Блок схема АЦП

Основні характеристики АЦП ATmega8:

- 10-бітова роздільна здатність;
- час перетворення 13÷260 мксек;
- 6 мультиплексованих вхідних каналів;
- діапазон вхідної напруги 0-Vcc АЦП;
- можливість вибору опорної напруги АЦП 2.56 В;
- режими перетворень (вільне виконання і окреме перетворення);
- виникнення переривання при завершенні перетворень;
- подавлення шуму в режимі сну.

АЦП має окремий штифт для зовнішнього джерело живлення $AV_{CC,}$ яке не має відрізнятися від V_{CC}. Є внутрішнє опорне джерело живлення 2.56 В.

АЦП перетворює аналогову вхідну напругу у цифрове 10-бітове значення методом послідовного наближення. Мінімальне значення відповідає GND, а максимальне значення відповідає значенню на штифті AREF мінус один наймолодший біт. До штифта AREF можна підключити AV_{CC} або 2.56 В. Внутрішня опорна напруга може бути розв'язана зовнішнім конденсатором на виводі AREF для покращення завадостійкість.

Аналоговий вхідний канал вибирається бітами MUX в ADMUX. Будь-які вхідні штифти АЦП, а також GND і опорна напруга з фіксованою забороненою зоною можуть бути обрані як одиночні входи в АЦП. АЦП вмикається встановленням біта ADEN в ADCSRA. Напруга опорна та вхідних каналів не стане чинною, доки не буде встановлено біт ADEN. АЦП не споживає електроенергію, коли ADEN очищений, тому рекомендується вимкнути АЦП перед переходом в режим енергозбереження сплячого режиму.

АЦП генерує 10-бітовий результат, який знаходиться у ріністрі даних ADC (ADCH і ADCL). За замовчуванням результат заданий із зсунути праворуч, але його можна за бажанням зсунути ліворуч, встановивши біт ADLAR в ADMUX.

Якщо результат зсунутий ліворуч і потрібна точність не більше ніж 8 біт, достатньо прочитати регістр ADCH. В іншому випадку спочатку потрібно прочитати ADCL, а потім ADCH, щоб переконатися, що вміст реєстрів даних належить до того самого перетворення. Після зчитування ADCL доступ АЦП до регістрів даних блокується. Це означає, що якщо ADCL було прочитано, і перетворення завершується до зчитування ADCH, жоден регістр не оновлюється, а результат перетворення втрачено. Коли ADCH зчитується, доступ АЦП до регістрів ADCH і ADCL знову активується.

АЦП має власне переривання, яке може викликатися після завершення перетворення. Коли доступ АЦП до регістрів даних заборонений між зчитуванням ADCH і ADCL, переривання спрацює, навіть якщо результат буде втрачено.

Окреме перетворення починається записом лог. 1 в ADSC. Цей біт залишається в лог. 1, поки триває перетворення, і буде апаратно очищеним, коли перетворення завершиться. Якщо під час перетворення вибрано інший канал даних, АЦП завершить поточне перетворення перед виконанням зміни каналу.

У режимі вільного запуску АЦП постійно відбирає та оновлює регісто даних ADC. Режим вільного запуску вибирається шляхом запису лог. 1 в біт ADFR регістра ADCSRA. Перше перетворення необхідно розпочати записом лог. 1 біт ADSC регістра ADCSRA. У цьому режимі АЦП виконуватиме послідовні перетворення незалежно від того, очищено прапор переривання ADIF чи ні.

За замовчуванням схема послідовного наближення вимагає вхідної тактової частоти від 50 кГц до 200 кГц для отримання максимальної роздільної здатності. Якщо потрібна менша роздільна здатність, ніж 10 біт, вхідна тактова частота АЦП може бути вище 200 кГц, щоб отримати вищу частоту дискретизації.

1.2. Регістри стану і керування

Регістр ADMUX (ADC Multiplexer Selection Register) – регістр керування мультиплексором

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

• REFS1, FERSS0 (Reference Selection Bits) – біти вибору опорного джерела для АЦП.

Якщо ці біти змінити під час перетворення — зміна напруги відбудеться тільки при наступному перетворенні.

REFS1	REFS0	Вибір джерела опорної напруги
0	0	AREF, внутрішній Vref відключений
1	0	AVCC із зовнішнім конденсатором на виводі AREF
1	0	Резерв
1	1	Внутрішнє джерело опорної напруги 2,56 В на виводі AREF

• ADLAR (ADC Left Adjust Result) – відповідає за подання результату АЦП в регістрі даних. Вствновлення в лог. 1 приводить до негайного зсуву результату перетворень вліво (для поточного і наступних перетворень), встановлення в лог. 0 приводить до зсуву вправо.

• MUX4,…,MUX0 (Analog Channel and Gain Selection Bits) – вибір аналогових каналів і коефіцієнтів підсилення.

Одиночні канали: 00000 – ADC0, 00001 – ADC1, 00010 – ADC2, 00011 – ADC3, 00100 – ADC4, 00101 – ADC5, 00110 – ADC6, 00111 – ADC7.

Диференціальні канали:

	Диф.вхід+	Диф. вхід-	Підсилення		Диф.вхід+	Диф. вхід-	Підсилення
01000	ADC0	ADC0	10X		...		
01001	ADC1	ADC0	10X	10100	ADC4	ADC1	1X
01010	ADC0	ADC0	200X	10101	ADC5	ADC1	1X
01011	ADC1	ADC0	200X	10110	ADC6	ADC1	1X
01100	ADC2	ADC2	10X	10111	ADC7	ADC1	1X
01101	ADC3	ADC2	10X	11000	ADC0	ADC2	1X
01110	ADC2	ADC2	200X	11101	ADC5	ADC2	1X
01111	ADC3	ADC2	200X	11110	1,22V (BG)	ADC2	1X
10000	ADC0	ADC1	1X	11111	0V(GND)		

Після завершення перетворень (ADIF лог. 1) результат знаходиться у регістрі ADC. Для одиночного перетворення результат наступний:

$$ADC = V_{IN} \cdot 1024 / V_{REF},$$

де V_{IN} – напруга на вибраному вхідному штифті;

V_{REF} – вибрана опорна напруга. 0x000 відповідає землі, 0x3FF відповідає вибраній опорній напрузі мінус мінус один найменш значимий біт.

Регістр контролю та стану АЦП **ADCSRA**:

ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN – лог. 1 включення, лог. 0 виключення поточного перетворення АЦП.

ADSC – лог. 1 в режимі окреме перетворення запускає кожне перетворення. У режимі вільного виконання запускає перше перетворення. По завершенню перетворення скидається в лог. 0.

ADFR – лог. 1 запуск, лог. 0 – зупинка режиму вільного виконання. У режимі вільного виконання АЦП робить вибірки і оновлює регістр даних неперервно.

ADIF – прапор переривання. Встановлюється в лог. 1 коли перетворення закінчено і регістр даних оновлено.

ADIE – дозвіл переривання АЦП. При лог. 1 і встановленні прапора I в SREG, то дозволені переривання завершення перетворень в АЦП.

ADPS2, ADPS1, ADPS0 – задають коефіцієнти дільника:

ADPS2	ADPS1	ADPS0	Дільник
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регістр даних **ADC** (8-розрядний) – містить результати перетворень.

ADLR=0

-	-	-	-	-	-	ADC9	ADC8	ADCH
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL

ADLR=1

ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	-	-	-	-	-	-	ADCL

2. Практична частина

На рис. 3 МК Atmega8 зчитує 10 кОм потенціометр, підключений до PORT C, і вмикає (або вимикає) світлодіоди, підключені до PORT D. Тут АЦП використовується в режимі одиничного перетворення з 10-бітовою точністю. При зміні значення потенціометра програма перевіряє результат перетворення. Якщо значення менше 512, то засвітиться LED1 (на PD6), інакше LED2 (на PD7).

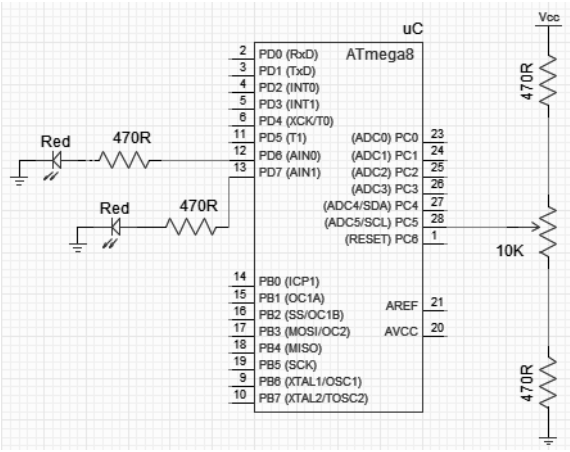


Рисунок 3 – Режим одиночного перетворення АЦП

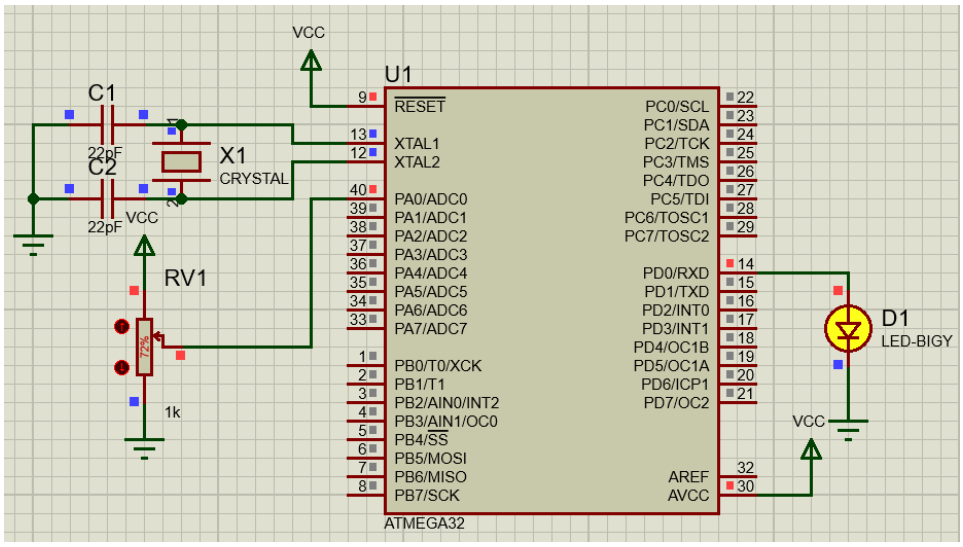


Рисунок 4 – Режим неперервного перетворення АЦП

Для налаштування АЦП (рис. 4) необхідно виконати наступні кроки:

1. Задати АЦП значення.

```
uint8_t ADCValue =128;
```

2. Сконфігурувати вихідний світлодіод.

```
DDRB|= (1 << PD0);
```

3. Сконфігурувати налаштування АЦП.

Це робиться шляхом встановлення бітів в регістрах керування АЦП. Спочатку встановлюється дільник для АЦП. Згідно з даними, дільник потрібно налаштувати так, щоб вхідна частота АЦП була між 50 кГц і 200 кГц. Тактовий сигнал АЦП є похідним від системного тактового сигналу. При системній частоті 1 МГц попередній дільник 8 дає частоту АЦП 125 кГц. Попередній дільник встановлюється бітами ADPS в регістрі ADCSRA. Згідно з даними, всі три біти ADPS2:0 мають бути встановлені в 011, щоб отримати дільник 8.

```
ADCSRA |= (0 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

Далі встановлюється опорна напругу АЦП. Це контролюється бітами REFS в регістрі ADMUX. Наступний код встановлює опорну напругу на AVCC:

```
ADMUX |= (1 << REFS0);
```

Щоб налаштувати канал мультиплексора потрібно задати біти MUX в регістрі ADMUX. Оскільки використовується ADC0 то:

```
ADMUX&=0xF0; ADMUX|=0;
```

Щоб перевести АЦП у режим вільного запуску, встановлюється в регістрі ADCSRA відповідний біт ADFR:

```
ADCSRA |= (1 << ADFR);
```

Щоб спростити зчитування значення АЦП, буде зроблено останню зміну налаштувань. Хоча АЦП має роздільну здатність 10 біт, ця кількість інформації часто не потрібна. Це 10-бітове значення розбивається на два 8-розрядні регістри, ADCH і ADCL. За замовчуванням наймолодші 8 біт значення ADC знаходяться в ADCL, а два старших є двома молодшими бітами ADCH. Встановивши біт ADLAR в регістрі ADMUX, можна вирівняти значення АЦП ліворуч. Це поміщає старші 8 біт вимірювання в регістр ADCH, а решту в регістр ADCL. Якщо потім прочитати регістр ADCH, то отримується 8-розрядне значення, яке задає результат вимірювання від 0 до 5 вольт у вигляді числа від 0 до 255. Якщо прийнятна невисока точність перетворень то можна замість 10-розрядного вимірювання прийняти 8-розрядне. Код для встановлення біта ADLAR:

```
ADMUX |= (1 << ADLAR);
```

4. Надати дозвіл на АЦП перетворення.

Для надання дозволу на АЦП перетворення встановляється біт ADEN у регістрі ADCSRA:

```
ADCSRA |= (1 << ADEN);
```

5. Запустити АЦП перетворення.

Для запуску перетворень потрібно встановити біт ADSC в регістрі ADCSRA:

```
ADCSRA |= (1 << ADSC);
```

У цей момент АЦП почне безперервно відбирати напругу, яка подається штифт ADC5. Код до цього моменту буде виглядати так:

6. Встановити нескінченний цикл `while`.

Потрібно перевірити значення регістра ADC і налаштувати світлодіоди на відображення високого/низького рівня. Оскільки показання ADC в ADCH має максимальне значення 255, було вибрано тестове значення 128, щоб визначити, чи була напруга є високою чи низькою. Простий оператор IF/ELSE в циклі WHILE дозволить увімкнути правильний світлодіод:

```
if(ADCH >ADCValue)
{
    PORTB |= (1 << PD0); // Turn on LED
}
else
{
    PORTB&= ~(1 << PD0); // Turn off LED
}
```

Приклад 1.

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

```
int main (void)
{
uint8_t ADCValue = 128;
DDRB |= (1 << PD0); // встановити LED1 на вихід
// Set ADC prescalar to 8
// 125KHz частота вибірки 1MHz
ADCSRA |= (0 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
ADMUX |= (1 << REFS0); // встановити вибір опорної напруги AVCC
ADMUX |= (1 << ADLAR); // Зсув ліворуч регістра ADC для отримання 8 біт
ADMUX&=0xF0;
ADMUX|=0; // вибір каналу АЦП ADC0
ADCSRA |= (1 << ADFR); // режим неперервних перетворень
ADCSRA |= (1 << ADEN); // Дозвіл АЦП перетворень
ADCSRA |= (1 << ADSC); // Запуск АЦП перетворень
while(1) // нескінченний цикл
{
if(ADCH > ADCValue)
{
   PORTB |= (1 << PD0);  // включення LED1
}
else
{
   PORTE &= ~(1 << PD0); // виключення LED1
}
}
```

Приклад 2.

```
/*
Програма отримує значення АЦП з входу ADC0
і відображає його значення на 7-сегментних індикаторах.
Значення напруги на ADC0 можна змінювати потенціометром
---------------------------------
*/
// Робоча частота МК (для delay.h)
#define F_CPU 8000000

#include <AVR/io.h>
#include <AVR/interrupt.h>
#include <Util/delay.h>

// Константи визначають індикатори
#define DIS3 0x08
#define DIS2 0x04
#define DIS1 0x02
#define DIS0 0x01
#define NO_DIS 0x00

// Константи кодують символи від 0 до 9
// (для виведення на 7-сегментні індикатори)
#define SYM_0 0x3F // "0"
#define SYM_1 0x06 // "1"
#define SYM_2 0x5B // "2"
#define SYM_3 0x4F // "3"
#define SYM_4 0x66 // "4"
#define SYM_5 0x6D // "5"
```

```
#define SYM_6 0x7D // "6"
#define SYM_7 0x07 // "7"
#define SYM_8 0x7F // "8"
#define SYM_9 0x6F // "9"

// Масив констант містить символи, які відповідають цифрам
const unsigned char SYM_ARRAY[] =
{SYM_0, SYM_1, SYM_2, SYM_3, SYM_4, SYM_5, SYM_6 ,SYM_7, SYM_8, SYM_9};

// Затримка між перемиканням символів для 7-сегментних індикаторів
#define SWITCH_DELAY 100 // (мкс)

// Глобальна змінна містить результат АЦП для ADC0
volatile unsigned char ADC0_Result = 0;


// Переривання при закінченні АЦП

ISR (ADC_vect)
{
// читання результату АЦП
ADC0_Result = ADCH;

// Запуск наступного АЦП
ADCSRA |= 0b01000000;
}

//
// Підпрограма налаштування АЦП
//
void ADC_Init (void)
{
// Опорна напруга - AVCC = +5В
// Результат перетворення вирівняти ліворуч
// Перетворення виконати з ADC0 (PA0) входу
ADMUX = 0b01100000;

// Включення АЦП
// Дозвіл переривання при завершенні перетворення
// Робоча частота АЦП = Робоча частота МК / 128
ADCSRA = 0b10001111;
}

//
// Функція перетворює цифру в символ і виводить його
// у заданому індикаторі
//
void UpdateDigit (unsigned char digitPosition, unsigned char digitValue)
{
// Перетворення цифри в символ
unsigned char symbolValue = SYM_ARRAY[digitValue];

// Вимикання всіх індикаторів
PORTB = NO_DIS;
```

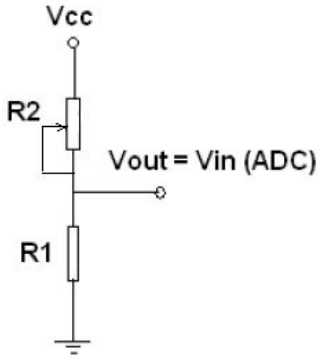
```
// Підготовка символу до виведення
PORTC = symbolValue;

// Включення відповідного індикатора
PORTB = digitPosition;

// Затримка (мкс)
_delay_us (SWITCH_DELAY);
}

// Виведення значення АЦП на 7-сегменті індикатори
//
void Update_7segmentDisplay (unsigned char data)
{
// Змінні зберігають значення одиниць, десятків і сотень
unsigned char units, decades, hundreds;

// Розбиття результату на окремі числа
hundreds = data / 100;
decades = (data - hundreds * 100) / 10;
units = data - hundreds * 100 - decades * 10;
// Виведення числа на індикатори
UpdateDigit (DIS0, units);
UpdateDigit (DIS1, decades);
UpdateDigit (DIS2, hundreds);
}

// Основна програма
int main (void)
{
// Налаштування портів введення/виведення
DDRB = 0x0F; // Керування 7-сегментними індикаторами (вибір)
DDRC = 0xFF; // Керування 7-сегментними індикаторами (дані)

// Налаштування ADC
ADC_Init();

// Дозвіл переривань
sei();

// Запуск АЦП
ADCSRA |= 0b01000000;
while (1)
{
// Виведення поточного значення значення АЦП на 7-сегментні індикатори
Update_7segmentDisplay (ADC0_Result);
}
}
```

Завдання.

1. Розробити і промоделювати у Proteus схеми цифрового вольтметра. Виміряну напругу виводити на 7-сегментний 4-розрядний цифровий світлодіодний індикатор.

2. Розробити схему і написати програму для вимірювання опору змінного резистора в схемі дільника напруги. Використати режим одиничного каналу, ADLAR=1.

ADCH = Vin * 256 / Vref.



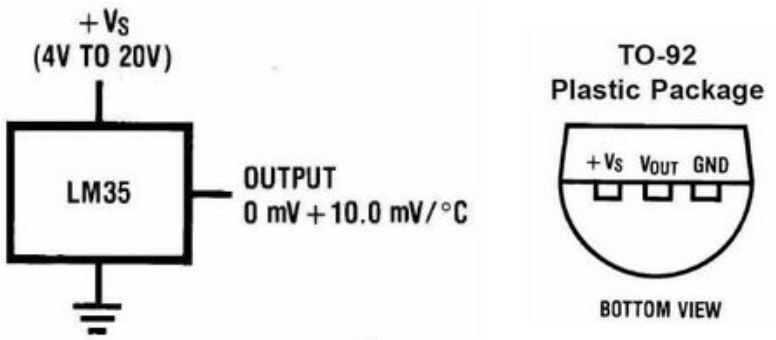
3. Розробити схему і написати програму для вимірювання температури з давача LM35.
Vout = T[°C]*0.01[V]/[°C].

Режим одиничного каналу ADC = $\frac{V_{INPUT}1024}{V_{REF}}$.

if ADLAR = 1 (low resolution): Vin = Vout, ADCH = Vin * 256 / Vref
ADCH = Vout * 256 / Vref
ADCH = T*2,56/Vref
if Vref = 2,56 V (внутрішнє джерело опорної напруги) ADCH = T [°C].

3.1. Схема включення і особливості давача.

Базова схема включення давача LM35 при вимірюванні температури в діапазоні +2...+150 °C.



На виході давача формується напруга пропорційна температурі по шкалі Цельсія, 10 мВ на 1 °C. Так, якщо температура давача 25 °C, то на його виході буде 250 мВ.

Питання.

1. Основні параметри АЦП.
2. Структурна схема АЦП.
3. Регістри АЦП.
4. Скільки каналів має АЦП.
5. Режими перетворення АЦП.
6. Як перетворюється вхідний сигнал у двійковий код?
7. Як програмно задається канал вимірювання аналогового сигналу.
8. Як задається точність вимірювання.
9. Джерела опорної напруги.
10. В яких одиницях відбувається вимірювання аналогового сигналу.

ЛАБОРАТОРНА РОБОТА № 14. Цифро-аналогові перетворювачі

Мета. Вивчення цифро-аналогових перетворювачів на основі МК.

1. Теоретична частина

1.1. Цифро-аналоговий перетворювач

Цифро-аналоговий перетворювач (ЦАП, Digit to Analog Converter, DAC) приймає цифрове значення і перетворює його у квазі-аналогову напругу. Рис. 1. Цифровий вхід може бути як послідовний, так і паралельний.

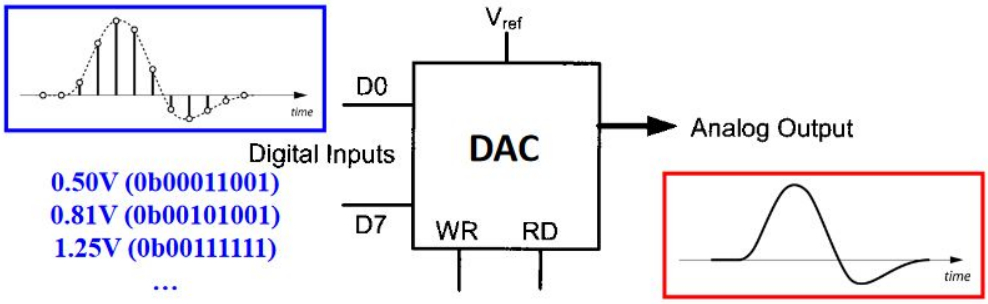


Рисунок 1 – Перетворення цифрового входу в аналоговий вихід

Найбільш поширені 8-, 10- і 12-розрядні входи, які дозволяють задати 256, 1024, 4096 дискретних рівнів аналогової напруги відповідно, рис. 2

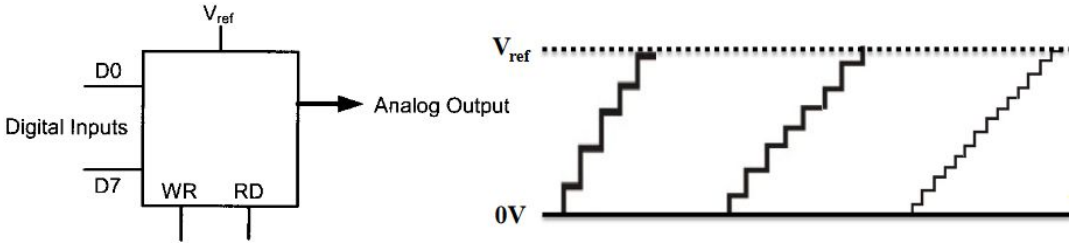


Рисунок 2 – Дискретні рівні аналогової напруги

У найпростішому випадкові ЦАП може бути реалізований за схемою суматора струмів, наприклад, на основі диференціального підсилювача в інвертуючому ввімкненні, на інверсному вході якого відбувається складання струмів, рис. 3. Зважаючи на те, що при заземленому неінверсному вході операційний підсилювач, охоплений негативним зворотним зв'язком, буде підтримувати нульовий потенціал і на інверсному вході, вхідні струми будуть однозначно визначатись вхідними напругами та опорами вхідних резисторів. Тим часом струм у ланцюзі зворотнього зв'язку визначається вихідною напругою та опором резистора зворотнього зв'язку. Позаяк сума вхідних струмів буде дорівнювати струму зворотнього зв'язку, вихідна напруга буде пропорційною сумі вхідних струмів.

Схема ЦАП із значеннями резисторів 2^nR, n=0,1,2,… рідко використовується при розрядності входу більше 6. Це зумовлено складністю технологічно забезпечити точність резисторів для великих значень n.

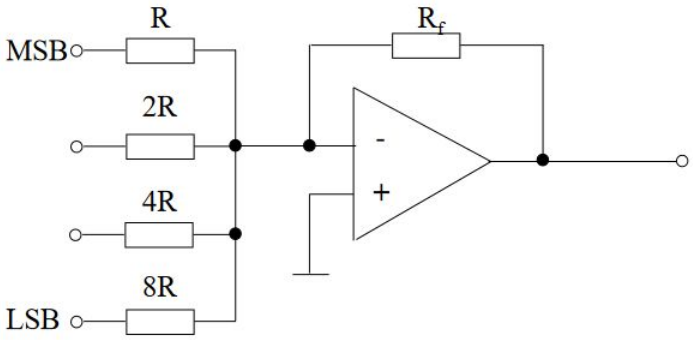


Рисунок 3 – Реалізація ЦАП на операційному підсилювачі

Більше поширення набула схема з'єднання резисторів R-2R, рис. 4.

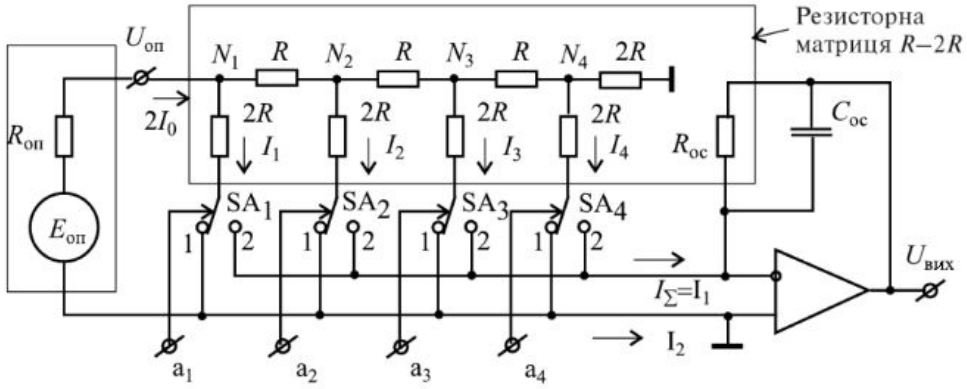


Рисунок 4 – Паралельний ЦАП на матриці R-2R

Сходинковий резистивний подільник R-2R складений таким чином, що передача розрядних напруг від точки N_j до точки N_{j+1} здійснюється з коефіцієнтом 0.5. У цьому неважко переконатися згортаючи і розгортаючи еквівалентну схему матриці, рис. 4А, складену із умови, що на інвертуючому вході ОП є віртуальний нуль.

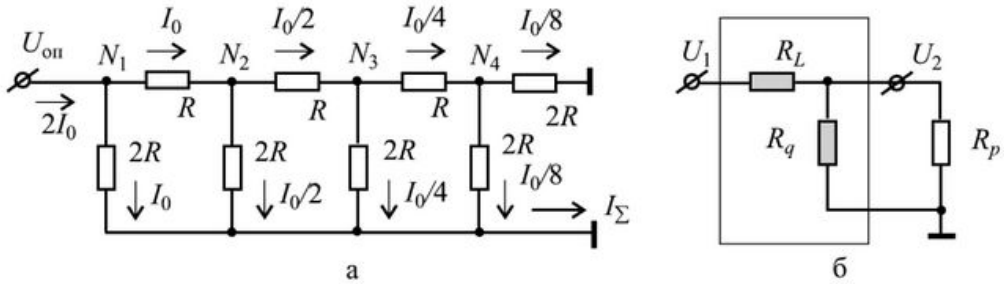


Рисунок 5 – Еквівалентні схеми: матриці а – R-2R; б – ланки ланцюгового подільника

Таким чином потенціали у точках Nj і розрядні струми Ij виявляються розподіленити за двійковим законом

$$I_j = \frac{U_{оп}}{2^{j-1} \cdot 2R}$$

В результаті нескладних перетворень можна показати, що вихідна напруга становить

$$U_{вих} = -R_{зз} I_{\Sigma} = -R_{зз} \sum_{j=1}^{n} I_j \cdot a_j = -R_{зз} \sum_{j=1}^{n} a_j \cdot \frac{U_{оп}}{2^{j-1} \cdot 2R} = -\frac{R_{зз} \cdot U_{зз}}{R \cdot 2^n} N \quad (1)$$

Таким чином, вихідна напруга лінійно залежить від коду N. Змінюючи $R_{зз}$, можна змінити масштаб перетворень. Струмовий вихід $I_{\Sigma}=I_1$ є нестабільним з витікаючим струмом, що унеможливлює використання однополярних ОП з позитивною напругою живлення.

Матриця R-2R є окремим випадком сходинкового або ланцюгового подільника, основною ланкою якого є Г-подібна структура з резисторів R_L і R_q, яка навантажена на опір навантаження R_p (рис. 5б). При синтезі ланки подільника виходять з того, що ланка повинна задовільняти двом умовам. По перше, умова нарощуваності і, по друге, умова фіксованого коефіцієнту передачі від ланки до ланки. Перша умова зводиться до того, що розподіл потенціалів у вузлах матриці не повинен мінятися при приєднанні додаткових розрядів, тобто вхідний опір ланки має бути постійним і рівним прикінцевому опору матриці R_p:

$$R_{вх} = R_L + (R_q паралель . R_p) = R_p = const \quad (2)$$

Друга умова:

$$\alpha = \frac{U_2}{U_1} = \frac{(R_q паралель . R_p)}{R_L + (R_q паралель . R_p)} = const \quad (3)$$

Розв’язуючи (2) і (3) відносно R_L і R_p , отримаємо:

$$R_L = \frac{(1-\alpha)^2}{\alpha} R_q. \quad (4)$$

$$R_p = \frac{(1-\alpha)}{\alpha} R_q \quad (5)$$

Задавши у виразах (4), (5) α=0.5, R_q=2R, отримаємо:

$$R_L = R, \; R_p = 2R.$$

Позитивною властивістю даного ЦАП є те, що сумарний опір матриці росте за лінійним законом від числа розрядів:

$$R_{\Sigma} = 2Rn + 2R + R(n-1) = R \cdot (3n+1).$$

1.1. ЦАП в мікроконтролерах AVR

ЦАП перетворює цифрове значення, записане в регістр даних (DACn.DATA), в аналогову напругу. Діапазон перетворення знаходиться між GND і вибраною опорною напругою в периферійному пристрої опорної напруги (VREF). ЦАП має один безперервний часовий вихід з високими керуючими можливостями. Перетворення DAC можна запустити з програми шляхом запису в регістр даних (DACn.DATA). Блок схема ЦАП показана на рис. 6.

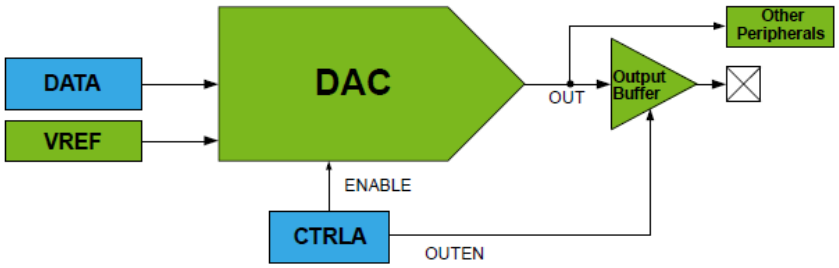


Рисунок 6 – Блок схема ЦАП МК AVR 32/64/128DBx

Ініціалізація ЦАП:

1. Вибрати опорну напругу ЦАП у периферійному пристрої опорної напруги (VREF), написавши відповідне біти вибору посилання.

2. Налаштувати подальше використання виходу ЦАП:

– налаштувати внутрішній периферійний пристрій для використання виходу ЦАП. Звернутися до відповідної документації периферійного пристрою.

– встановити контакт на вихід, записавши «1» у біт вихідного буфера (OUTEN). Вхід для контакту ЦАП має бути вимкнено на периферійному порту (ISC = INPUT_DISABLE у PORTx.PINCTRLn).

3. Записати початкове цифрове значення в регістр даних (DACn.DATA).

4. Увімкнути ЦАП, записавши «1» в біт ENABLE в регістрі Control A (DACn.CTRLA).

Робота ЦАП:

1. Дозвіл, заборона, скидування.

DAC вмикається шляхом запису «1» в біт ENABLE в регістрі Control A (DACn.CTRLA), а вимикається за допомогою запис «0» до цього біта.

2. Запуск перетворення.

Коли в біт ENABLE у регістрі Control A (DACn.CTRLA) записується «1», перетворення починається, як тільки регістр даних (DACn.DATA) відбувається запис.

Коли в біт ENABLE у регістрі DACn.CTRLA записується «0», запис у регістр даних не запускає перетворення. Натомість перетворення починається, коли біт ENABLE у DACn.CTRLA записується в «1».

3. ЦАП як джерело для внутрішньої перефирії.

Аналоговий вихід ЦАП може бути внутрішньо підключений до інших периферійних пристроїв, коли в біт ENABLE в Control A (DACn.CTRLA) регістрі записується «1». Коли аналоговий вихід ЦАП використовується лише внутрішньо, вихідний буфер (OUTEN) у DACn.CTRLA може бути «0».

4. Вихід ЦАП на контакт.

Аналоговий вихід ЦАП можна підключити до контакту, записавши «1» у біт дозволу вихідного буфера (OUTEN) у регістрі керування A (DACn.CTRLA). Вивід, який використовується ЦАП, повинен має мати заборонений вхід із периферійного порту. Між виходом ЦАП і контактом є вихідний буфер, який гарантує незалежність аналогового значення від навантаження контакту. Вихідний буфер може бути лише джерелом струму і має дуже обмежені можливості споживання.

5. Робота в режимі Sleep.

Якщо в біт Run in Standby біті (RUNSTDBY) у регістрі Control A (DACn.CTRLA) записано «1», ЦАП продовжить працювати в режимі Sleep (очікування). Якщо біт RUNSTDBY дорівнює нулю, ЦАП зупинить перетворення в режимі Sleep.

Якщо перетворення зупинено в режимі очікування, ЦАП і вихідний буфер вимикаються, щоб зменшити споживання енергії. Коли пристрій виходить із режиму очікування, ЦАП і вихідний буфер (якщо біт OUTEN у регістрі Control A (DACn.CTRLA) записаний як «1») знову вмикаються. Таким чином, потрібен час запуску перед початком нового перетворення.

У режимі очікування Power-Down ЦАП і вихідний буфер вимикаються, щоб зменшити енергоспоживання.

Регістри ЦАП:

			7	6	5	4	3	2	1	0
0x00	CTLA	7:0	RUNSTBBY	OUTEN						Enable
0x01	Reserved									
0x02	DATA	7:0	DATA[1:0]							
		15:8	DATA[9:2]							

Біт 7 – RUNSTDBY Запуск у режимі очікування Якщо в цей біт записано «1», ЦАП або вихідний буфер не будуть автоматично відключені, коли пристрій входить Режим очікування.

Біт 6 – увімкнення вихідного буфера OUTEN.Запис «1» у цей біт активує вихідний буфер і надсилає сигнал OUT на контакт.

Біт 0 – ENABLE Дозвіл ЦАП. Запис «1» в цей біт вмикає ЦАП.

Пара регістрів DACn.DATAL і DACn.DATAH задає 10-бітове значення DACn.DATA. Два біти LSbs [1:0] є доступними за початковим зміщенням. До восьми бітів MSbs [9:2] можна отримати доступ за зміщенням + 0x01. Вихідні дані буде оновлено після запису DACn.DATAH.

Біти 15:6 – DATA[9:0] Ці біти містять цифрові дані, які будуть перетворені в аналогову напругу.

2. Практична частина

В практичні частині потрібно розробити генератор сигналів заданої форми і частоти з використанням МК AVR128DB28.

Як приклад показана програма для генерації сигналів синусоїди, трикутників, прямокутників з використанням МК AVR128D28.

2.1 Конфігурація і ініціалізація ЦАП

Конфігурування ЦАП:

1. Вибір внутрішньої опорної напруги 4.096 В у регістрі DAC0 записом 0x2 у біти REFSEL[2:0].

Bits 2:0 – REFSEL[2:0] Reference Select
This bit field controls the reference voltage level for DAC0.

Value	Name	Description
0x0	1V024	Internal 1.024V reference(1)
0x1	2V048	Internal 2.048V reference(1)
0x2	4V096	Internal 4.096V reference(1)
0x3	2V500	Internal 2.500V reference(1)
0x4	-	Reserved
0x5	VDD	VDD as reference
0x6	VREFA	External reference from the VREFA pin
0x7	-	Reserved

Зауважимо, що час ввімкнення опорної напруги 50 мкс. Тому потрібна затримка на 50 мкс після дозволу на використання опорної напруги.

.....continued						
Symbol	**Description**	**Min.**	**Typ.**	**Max.**	**Units**	**Conditions**
V_{VREF_2V500}[2]	Internal Voltage Reference 2.5V	—	±4	—	%	V_{DD} ≥ 2.7V, -40°C to 85°C
T_{VREF_ST}	VREF start-up time	—	50	—	µs	

2. Виходом ЦАП є контакт PD6 у порті POTRD. Встановити контакт доступним на вихідних сигналів буфера записом ‘1’ у біт OUTEN регістра DAC.CNTRLA.

3. Заборонити вхідні сигнали контакту PD6 для портів периферії. Записати ‘0x04’ у біти ISC[2:0] для заборони вхідного буфера у POTRD.PIN6CNTL.

Bits 2:0 – ISC[2:0] Input/Sense Configuration

This bit field controls the input and sense configuration of pin n. The sense configuration determines how a port interrupt can be triggered.

Value	Name	Description
0x0	INTDISABLE	Interrupt disabled but digital input buffer enabled
0x1	BOTHEDGES	Interrupt enabled with sense on both edges
0x2	RISING	Interrupt enabled with sense on rising edge
0x3	FALLING	Interrupt enabled with sense on falling edge
0x4	INPUT_DISABLE	Interrupt and digital input buffer disabled[1]
0x5	LEVEL	Interrupt enabled with sense on low level
other	—	Reserved

4. Записати початкове значення у регістр DATA.

5. Дозволити роботу ЦАП записом ‘1’ у біт ENABLE регістра DAC.CTRLA.

Налаштування контактів.

З використанням ЦАП буде згенеровано сигнали синусоїди, трикутників і прямокутників із заповненістю циклу 50% і частотою 50 Гц. Контакт PD6 буде під’єднано до осцилографа для спостереження форми сигналу. Контакт PA6 буде використовуватися як зовнішнє переривання для перемикання на іншу форму сигналу. При натисканні кнопки, під’єднаної до PA6, CPU переходить до ISR і збільшує змінну, яка буде використовуватися для зміни форми сигналу.

Щоб використати PA6 як зовнішній контакт потрібно:

1. Встановити напрям даних на введення.
2. Увімкнути Pull-up резистори.
3. Увімкніть переривання за негативним фронтом сигналу.
4. Увімкнути глобальне переривання

Генерація синусоїди.

Перш ніж генерувати синусоїду, потрібно обчислити вибірки, що відповідають (1/50) періоду, і зберегти їх у буфері. Синус створюється за допомогою фіксованої кількості кроків, усі кроки виконуються за один період. Щоб отримати гарну форму, вибрано 250 вибірок. функція Для отримання цих значень sin() використано функцію sin(). Затримки між кожним кроком розраховується наступним чином:

$$fд = 1/fзк = fc \cdot N$$

fд — частота дискретизації;

fзк — затримка одного кроку;

N – число вибірок.

Генерація трикутних сигналів.

Використовується той самий буфер (на 250 кроків), щоб заповнити його значеннями, які поступово збільшуються, коли сигнал зростає (125 кроків), і значеннями, які зменшуються, коли сигнал спадає (125 кроків).

Генерація прямокутних сигналів.

Прямокутні сигнали легко отримати за допомогою ЦАП, оскільки є лише два стани: високий або низький. Записується максимальне значення 1023 до регістру даних, щоб отримати максимальну напругу 4,096 В, і записується мінімальне значення 0, щоб отримати 0 В

2.2. Код програми

```
// main.c
#include "dac.h"
#include <avr/io.h>
#include <avr/interrupt.h>

uint16_t my_buffer[250] = {0};
volatile uint8_t wave_select = 0;

ISR(PORTA_PORT_vect)
{
     _delay_ms(80);
     wave_select++;
     PORTA.INTFLAGS |= 1 << 6;//clear flag
}

int main(void)
{
     dac_init();
     port_init();
     uint8_t index;
     sei();
    for (;;)
    {
          switch(wave_select)
          {
               case 0:
               sine_wave_samples(my_buffer);

               for (index = 0; wave_select == 0; ++index)
               {
                    write_dac(my_buffer[index]);
                    _delay_us(STEP_DELAY);
                    if (index >= 250)
                    {
                         index = 0;
                    }
               }
               break;

               case 1:
               triangle_samples(my_buffer);

               for (index = 0; wave_select == 1; ++index)
               {
```

```
                    write_dac(my_buffer[index]);
                    _delay_us(STEP_DELAY);
                    if (index >= 250)
                    {
                         index = 0;
                    }
               }
               break;

               case 2:
               for (; wave_select == 2;)
               {
                    write_dac(512);
                    _delay_ms(9);
                    write_dac(0);
                    _delay_ms(9);
               }
               break;
          }

          if (wave_select >= 3)
          {
               wave_select = 0;
          }
     }
}

// dac.c
#include "dac.h"

void dac_init(void)
{
     VREF.DAC0REF = 0x02;//4.096V as reference
     VREF.DAC0REF |= 1 << 7;//Reference always on
     _delay_us(50);//Wait Start-up time

     //PD6 is DAC0 OUT
     PORTD.PIN6CTRL = 0x04;//Disable input, disable pull-up
     write_dac(128);//initial value
     DAC0.CTRLA |= 1 << 6 | 1 << 0;// Enable output pin and DAC0
}

void write_dac(uint16_t value)
{
     DAC0.DATA = value  << 6;
}

void sine_wave_samples(uint16_t simples[])
{
     for (uint8_t i = 0; i < 250; ++i)
     {
          simples[i] = lrintf(512 + 511 * sin((2 * M_PI * i) / 250));

     }
}
```

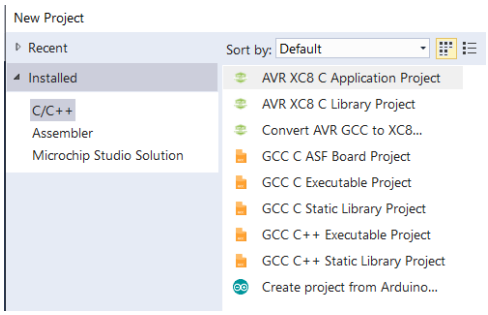
```
void triangle_samples(uint16_t values[])
{

     for (uint8_t i = 0; i < 125; ++i)
     {
          values[i] = 8 * i;
     }

     for (int16_t i = 125, j = 125; j < 250; --i, ++j)
     {
          values[j] = i * 8;
     }
     values[250] = 0;
}

void port_init(void)
{
     PORTA.DIRCLR |= 1 << 6;//PA6 input
     PORTA.PIN6CTRL = 0x03;//Falling edge trigger
     PORTA.PIN6CTRL |= 1 << 3;//Enable pull-up
}
```

// dac.h

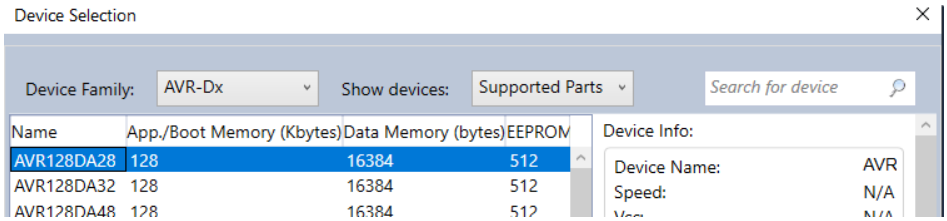
```
#ifndef DAC_H_
#define DAC_H_

#define F_CPU 4000000UL
#include <util/delay.h>
#include <avr/io.h>
#include <math.h>

#define STEP_DELAY 70

void write_dac(uint16_t);
void dac_init(void);
void sine_wave_samples(uint16_t simples[]);
void triangle_samples(uint16_t values[]);
void port_init(void);
#endif /* DAC_H_ */
```

2.3. Створення проекту в Microchip Studio і компіляція програми.

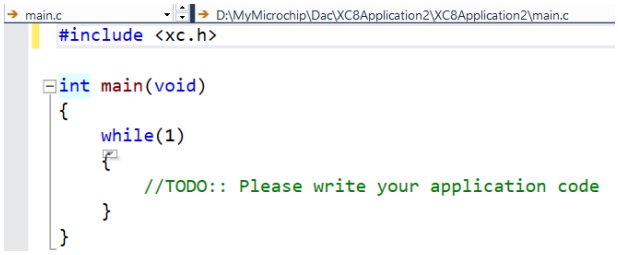
1. Створення нового проекту



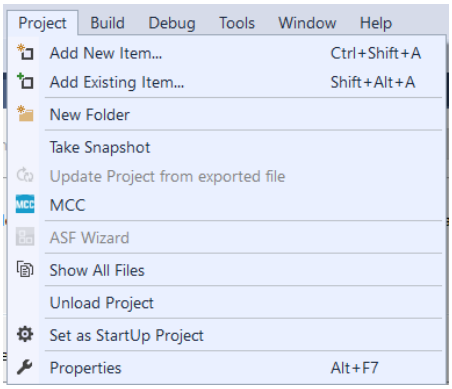
2. Вибір МК AVR128DA28



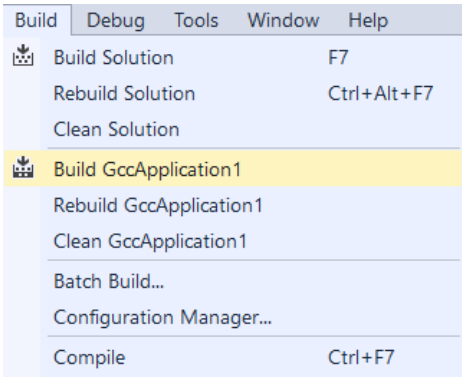
3. Створення головної функції main.c

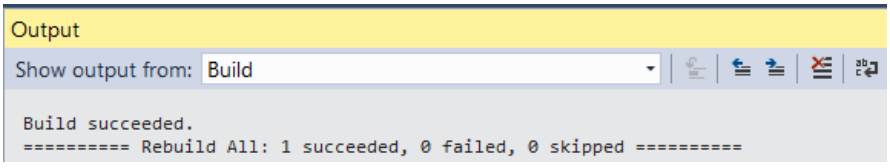


4. Добавлення в проект додаткових *.c і *.h файлів: Add Existing Item:



5. Компіляція проекту





6. Результат компіляції проекту.

Name	Date modified	Type	Size
dac.d	7/22/2022 4:19 PM	D File	4 KB
dac.o	7/22/2022 4:19 PM	O File	10 KB
GccApplication1.eep	7/22/2022 4:19 PM	EEP File	1 KB
GccApplication1.elf	7/22/2022 4:19 PM	ELF File	35 KB
GccApplication1.hex	7/22/2022 4:19 PM	HEX File	7 KB
GccApplication1.lss	7/22/2022 4:19 PM	LSS File	45 KB
GccApplication1.map	7/22/2022 4:19 PM	MAP File	43 KB
GccApplication1.srec	7/22/2022 4:19 PM	SREC File	7 KB
main.d	7/22/2022 4:19 PM	D File	4 KB
main.o	7/22/2022 4:19 PM	O File	10 KB
makedep.mk	7/22/2022 3:36 PM	MK File	1 KB
Makefile	7/22/2022 4:19 PM	File	5 KB

Завдання.

1. Для схеми рис. 8 отримати форму вихідного сигналу в режимі роботи генератора слова як наростаючого лічильника.

2. Для схеми рис. 8 записати потенціали у розрядних вузлах матриці R-2R в покроковому режимі роботи генератора слова.

3. Повторити пункти 2, 3 при значеннях резисторів 1 кОм і 2 кОм.

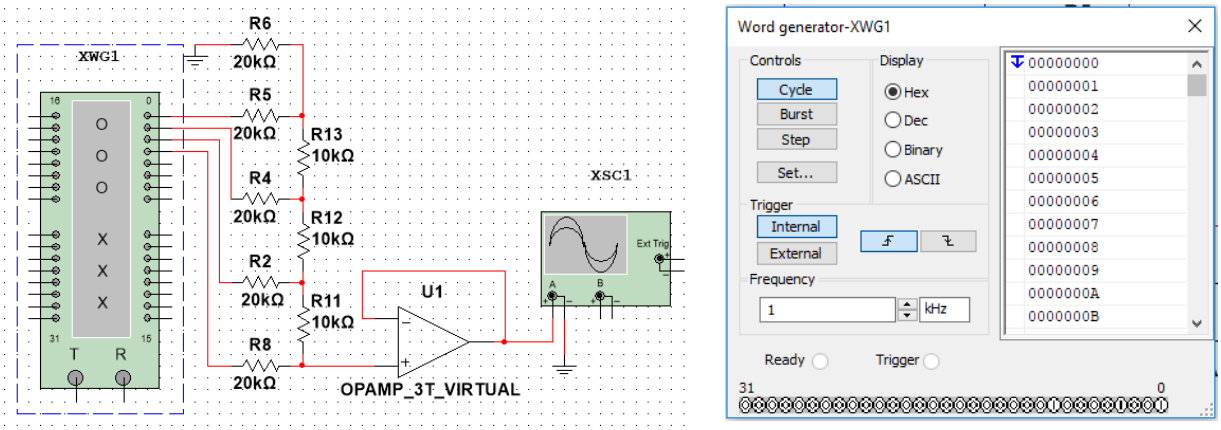


Рисунок 8 – Паралельний ЦАП на матриці R-2R з генератором слова

4. Для схеми на рис. 9 розрахувати за формулою вихідні значення ЦАП і порівняти їх з результатами моделювання.

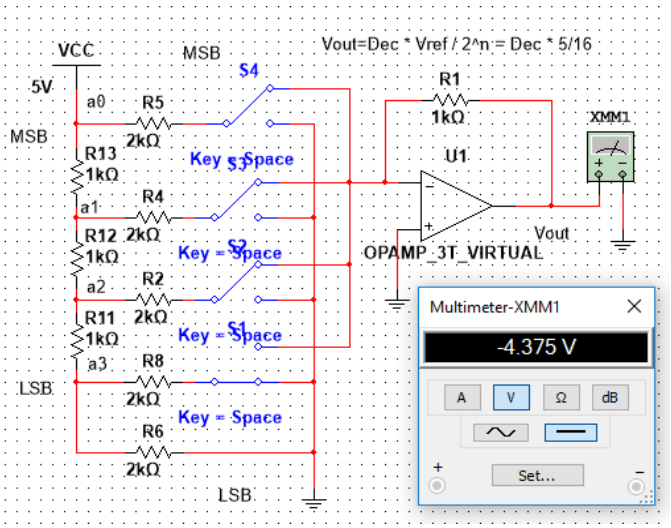


Рисунок 9 – Паралельний ЦАП на матриці R-2R з перемикачами

5. Для схеми на рис. 10 промоделювати вихідні значення ЦАП для значень резистора R1= 5 кОм, 10 кОм, 20 кОм. Пояснити отримані результати.

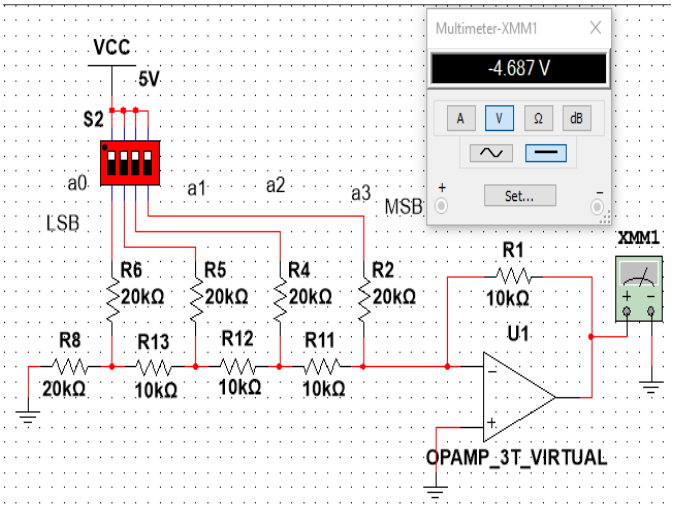


Рисунок 10 – Паралельний ЦАП на матриці R-2R з груповим перемикачем

6. Прошити МК AVR128DA28 hex файлом і продемонструвати на осцилографі форми згенерованих сигналів.

Запитання.

1. Яка роздільна здатність ЦАП?
2. Які джерела опорної напруги використовує ЦАП?
3. Регістри ЦАП.
4. Призначення регістра DCON.
5. Призначення регістра DAC.
6. Робота з 8- і 10-розрядним ЦАП.
7. Як перетворюється вхідний цифровий сигнал у вихідний аналоговий?

ЛАБОРАТОРНА РОБОТА № 15. Керування двигуном постійного струму

Мета. Вивчення керування роботою двигуна постійного струму за допомогою IC 555, L293D і мікроконтролера.

1. Теоретична частина

Двигун (мотор) постійного струму - це пристрій, який перетворює електричну енергію в механічну. Зокрема, мотор постійного струму використовує постійний струм для перетворення електричної енергії в механічну. Основним принципом роботи мотора є взаємодія між магнітним полем і струмом для створення сили всередині мотору, яка допомагає мотору обертатися. Отже, коли електричний струм проходить через котушку в магнітному полі, утворюється магнітна сила, яка створює крутний момент, що призводить до руху мотора. Мотор постійного струму невеликої потужності показаний на рис. 1

Рисунок 1 – Мотор постійного струму

Напрямок обертання мотора контролюється шляхом реверсування струму. Швидкість мотора можна змінювати, змінюючи його напругу живлення. Для керування швидкістю мотора можна використати схему ШІМ (наприклад IC 555) або мікроконтролери які мають вбудовані таймери/лічильники з підтримкою ШІМ.

Принцип керування швидкістю мотора за допомогою ШІМ показаний на рис. 2.

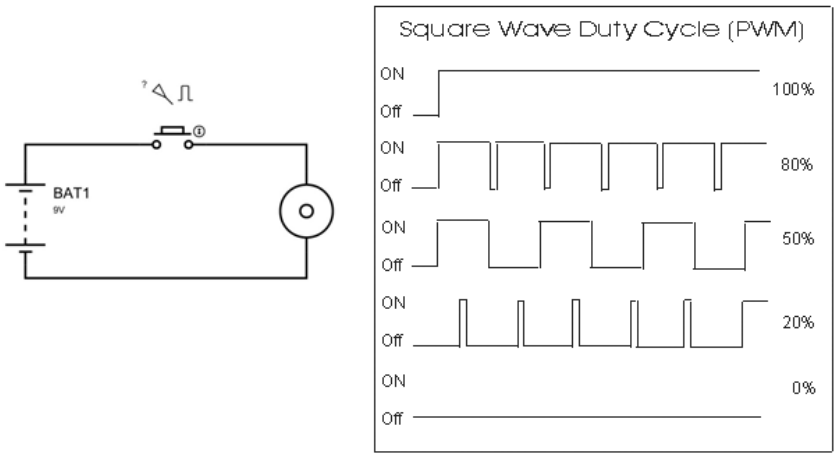


Рисунок 2 – Принцип керування швидкістю мотора за допомогою ШІМ сигналів

Для характеристики імпульсних сигналів використовуються безрозмірні величини коефіцієнт заповнення (D) і шпаруватість (S) або робочий цикл у відсотках D % (англ. duty cycle). Таким чином, для імпульсного сигналу:

$$D = \frac{\tau}{T},\ D = \frac{\tau}{T} \times 100\ \%,\ S = \frac{1}{D},$$

де D – коефіцієнт заповнення (або робочий цикл у %), τ – тривалість імпульсу, T – період імпульсів, S – шпаруватість;

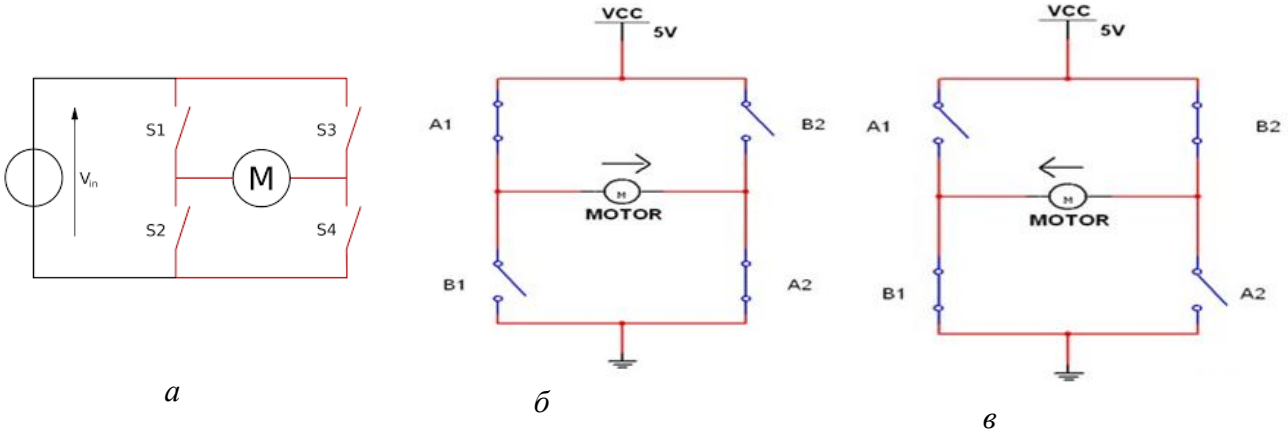
Приклад сигналу з коефіцієнтом заповнення 0,5 або 50% або шпаруватістю, рівною двом – меандр.

Якщо натиснути кнопку (рис. 2) то двигун почне обертатися і він буде рухатися, поки буде натиснута кнопка. Це натискання є безперервним і зображено на першій хвилі малюнка. Якщо кнопку натиснути 8 мс і відпустити на 2 мс за час циклу 10 мс, то середнє квадратне значення змінної напруги буду 7 В. Через це знижене значення діючої напруги двигун буде обертатися із зниженою швидкістю. Робочий цикл мотора становить 80% (8/(8+2)).

У другому та третьому випадках кнопка натискається ще менше часу, ніж у першому випадку. Через це діюча напруга на клемах двигуна ще більше зменшується. Завдяки такому зниженню напруги швидкість двигуна ще більше зменшується. Таким чином ШІМ можна використовувати для для зміни швидкості моторів.

1.2. Н-міст IC L293D

Для керування моторами малої потужності і зміни їх напрямку обертання використовується Н-міст, рис. 3. Два таких Н-мости реалізовані у IC L293D. Цю схему можна використовувати для керування двома двигунами постійного струму малої потужності з використанням сигналів від МК, рис. 4.



а *б* *в*

Рисунок 3 – Н-міст: *а*) схема; *б*) A1+A1 – обертання вперед; *в*) B1+B2 – обертання назад

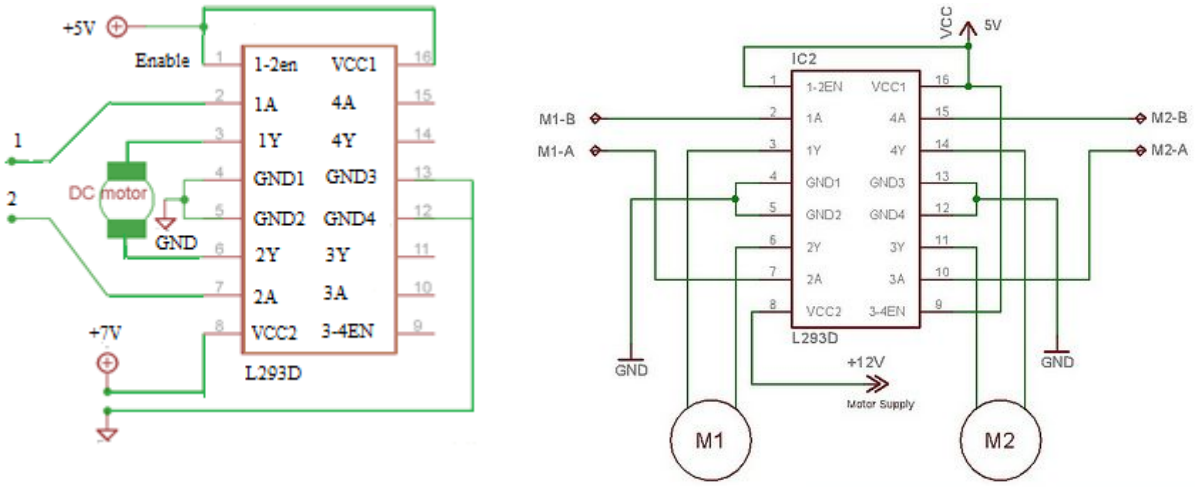


Рисунок 4 – IC L293D для керування DC мотором

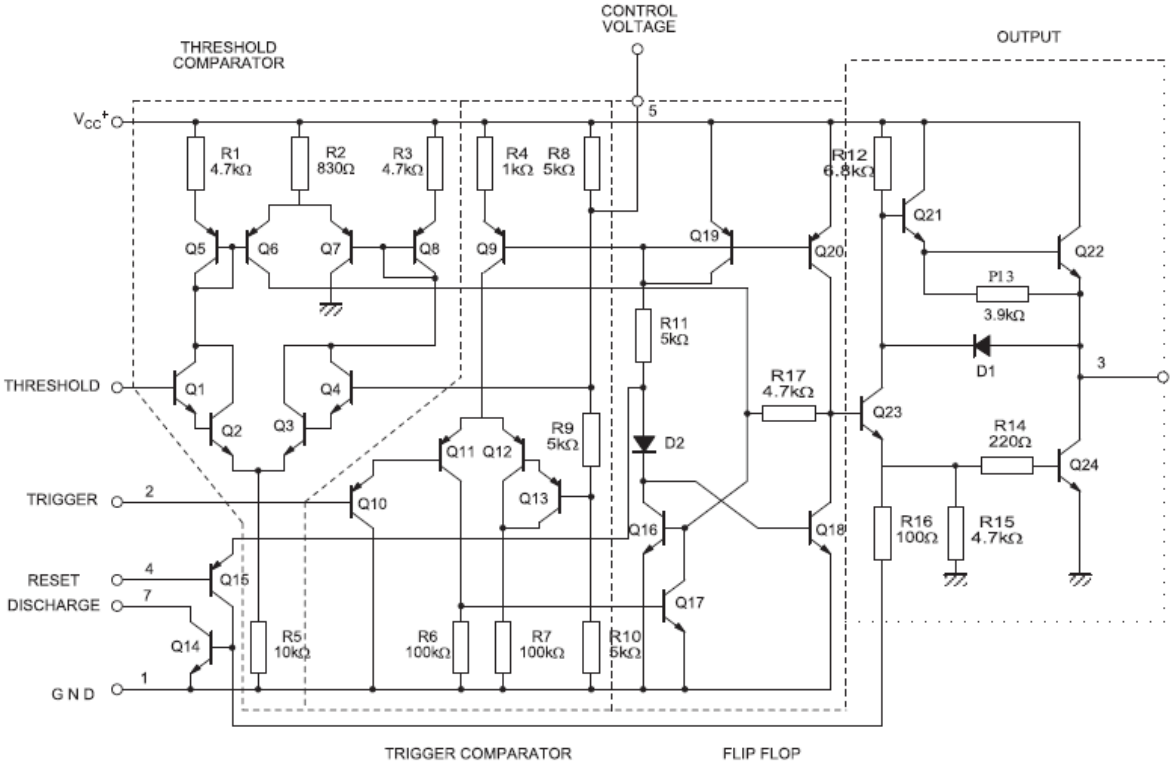
Напрям обертання мотора змінюється рівнями напруги на входах 1 і 2, табл. 1.

Таблиця 1 — Керування напрямком обертання мотора

Вхід 1	Вхід 2	Обертання
Низький	Високий	Вправо
Високий	Низький	Вліво
Низький	Низький	Стоп
Високий	Високий	Стоп

1.3. Таймер IC 555

Електричні схеми IC 555 різних виробників показані на рис. 5.



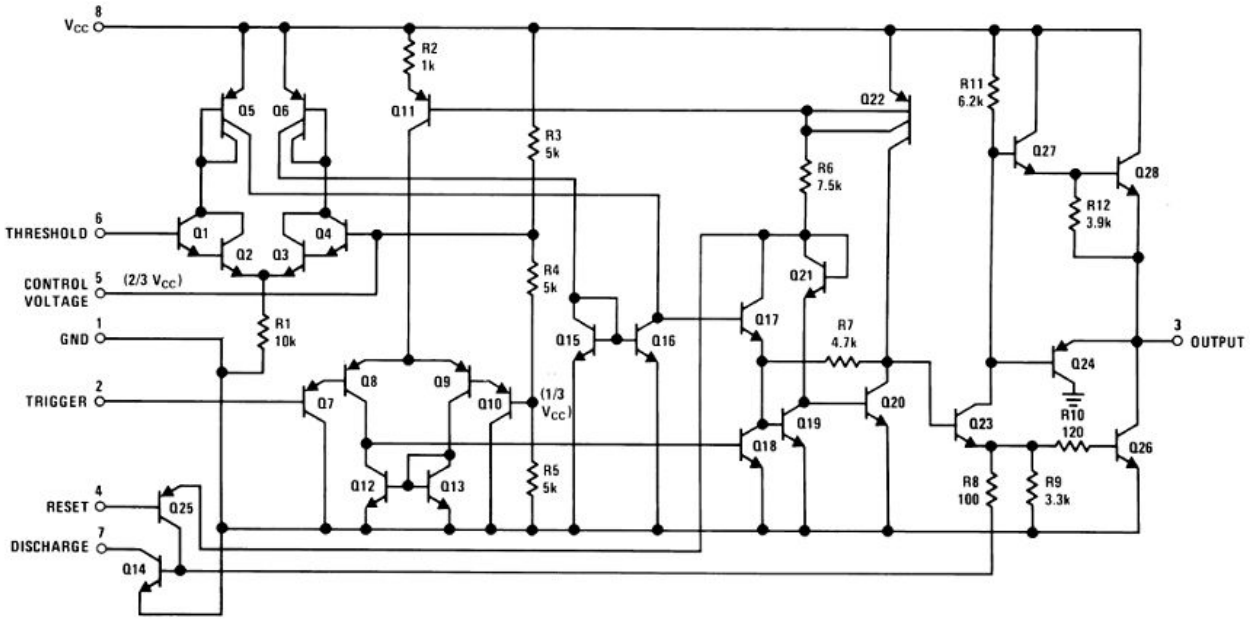


Рисунок 5 – Електричні схеми IC 555

Мікросхема дозволяє формувати одинокий імпульс або серію імпульсів зі стабільними тимчасовими характеристиками. Усередині кристалу IC 555 міститься близько 20 транзисторів, 15 резисторів та 2 діоди. Склад і кількість елементів можуть несуттєво змінюватися в залежності від виробника.

Умовно IC 555 складається з шести функціональних блоків, рис. 6:

- дільник напруги (1);
- два компаратори (2 і 3);
- RS-тригер (4);
- інвертуючий підсилювач потужності (5);
- транзистор з відкритим колектором на виході (6).

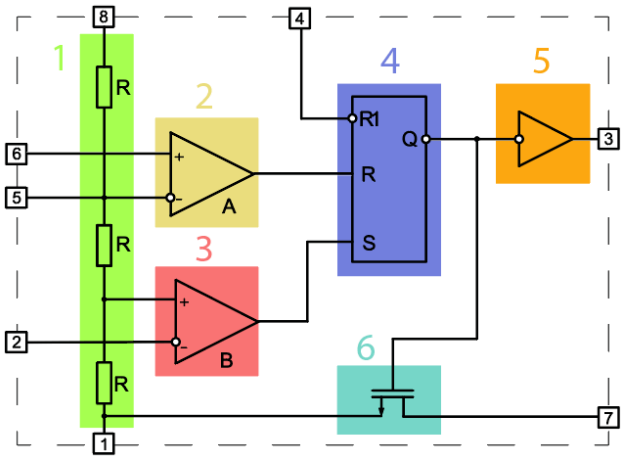


Рисунок 6 – Структурна схема IC 555

На вході знаходиться резистивний дільник напруги (1), який формує дві опорні напруги для компараторів (2 і 3). Вихідні сигнали компараторів надходять на RS-тригер (4) з зовнішнім виводом для скидання, а потім на підсилювач потужності (5). Останнім вузлом є транзистор з

відкритим колектором (6), який може виконувати кілька функцій, в залежності від поставленого завдання.

Мікросхема 555 та її аналоги випускаються в 8-контактному корпусі типу DIP-8, TSSOP або SOIC. Розташування виводів, незалежно від корпусу – стандартне, рис. 7.

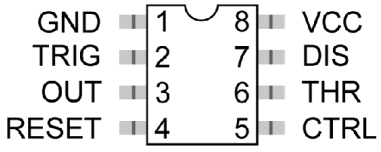


Рисунок 7 – Розташування і маркування виводів IC 555

Номер виводу	Позначення	Призначення	Опис
1	GND	Загальна земля	Вивід з'єднується із загальним дротом схеми чи мінусом джерела живлення
2	TRIG (trigger)	Запуск	Вхід компаратора «B». При надходженні на вхід «TRIG» сигналу довільної форми напругою менше ⅓ від «VCC», здійснюється запуск інтегрального таймера, внаслідок чого на виході «OUT» виникає логічна одиниця.
3	OUT	Вихід	Вихідний сигнал таймера. На вивід «OUT» формується одна з двох напруг, які приблизно відповідають рівням GND або VCC, в залежності від стану таймера.
4	RESET	Скидання	При надходженні на вихід «RESET» напруги живлення менш 0,7 В, відбувається скидання таймера і відповідно на виводі «OUT» з'явиться логічний нуль. Якщо в проектованій схемі не потрібен режим скидання, то бажано вивід «RESET» з'єднати з плюсом джерела живлення.
5	CTRL	Керування	Додаткове управління таймером. При подачі на вхід «CTRL» сигналу від 45 до 90% від «VCC», можна контролювати тривалість імпульсів на виході. Це дозволяє позбутися від зовнішнього RC ланцюжка. Якщо регулюються тимчасові параметри RC-ланцюжком, то необхідно підключити вивід «CTRL» до мінуса схеми через конденсатор 10 нФ.
6	THR (treshold)	Стоп	Вхід компаратора «A». При надходженні на вхід «THR» сигналу довільної форми напругою понад ⅔ від «VCC», зупиняється інтегральний таймер, внаслідок чого на виході «OUT» виникає логічний нуль.

7	DIS (dischard)	Розряд	Вихід «DIS» під'єднується до колектора вихідного транзистора. Використовується для розрядки часозадаючого конденсатора між інтервалами. Стани цього виходу повторюють стани основного виходу OUT, тому використовується для нарощування здатності навантаження таймера.
8	VCC	Живлення	Живлення мікросхеми. Вивід з'єднується з дротом живлення схеми з напругою від 4,5 до 16 вольт.

Таймер 555 підтримує три основні режими роботи:

- моностабільний режим (одновібратор);
- автоколивальний режим (мультивібратор);
- бістабільний режим (тригер Шмітта).

Схема підключення виводів в автоколивальному режимі показана на рис. 8.

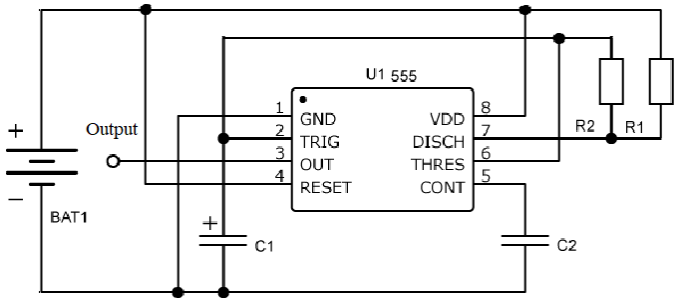


Рисунок 8 – Схема підключення виводів в автоколивальному режимі

Мікросхема видає послідовність прямокутних імпульсів на виході «OUT», параметри яких визначаються RC-ланцюжком з конденсатора *C1* та двох опорів: *R1* і *R2*.

$f = 1 / 0{,}693 * C1*(R1 + 2 \cdot R2) = 1.44 / (C1*(R1 + 2 \cdot R2))$

$T = 1 / f$

$T = t1 + t2 = 0{,}693 * C1 * (R1 + 2 \cdot R2)$

$t1 = 0{,}693 * C1 * (R1 + R2)$ - час зарядження (вихід ВИСОКИЙ)

$t2 = 0{,}693 * C1 * R2$ - час розрядження (вихід НИЗЬКИЙ),

$t1/(t1 + t2) = (R1 + R2)/(R1 + 2 \cdot R2)$ - коефіцієнт заповнення

де:

f – частота імпульсів, Гц;

T – період імпульсу, сек;

t1 – довжина імпульсу (логічна одиниця), сек;

t2 – довжина імпульсу (логічний нуль), сек;

R1, *R2* – опорів, Ом;

C1 – номінал ємності, Ф.

2. Практична частина

2.1. Автоколивальний режим (мультивібратор) таймера

Для реалізації схеми рис. 7 використовуються наступні компоненти:
• блок живлення +9 В;
• невеликий двигун змінного струму;
• таймер IC 555;
• опори 1кОм, 100 ом;
• h-міст L293D;
• змінний опір 100 кОм ÷ 220 кОм;
• діод IN4148 або IN4047×2;
• конденсатор 10 нФ або 22 нФ;
• перемикач.
Електрична схема керування мотором постійного струму показана на рис. 9.

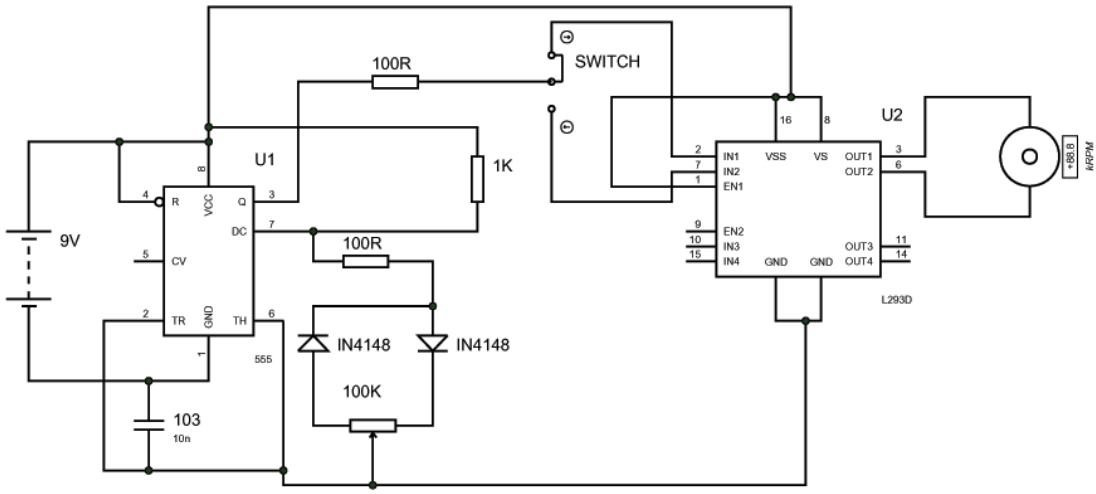


Рисунок 9 – Електрична схема керування мотором

Перемикач використовується для зміни напрямку обертання мотора. Змінний опір використовується для керування швидкістю обертання вала мотора. Таймер генерує ШІМ сигнал із шпаруватістю, яка визначається величинами постійного і змінного опору та конденсатора. Так як зарядження і розрядження конденсатора відбувається через різні діодні гілки, то і час вихідного сигналу у високому і низькому рівні буде різний (тобто буде формуватися ШІМ сигнал). ШІМ сигнал подається на Н-міст для керування мотором.

2.2. Керування двигуном постійного струму за допомогою МК

Приклад 1. Керування напрямом обертання мотора за допомогою МК ATmega 32A, рис. 10.

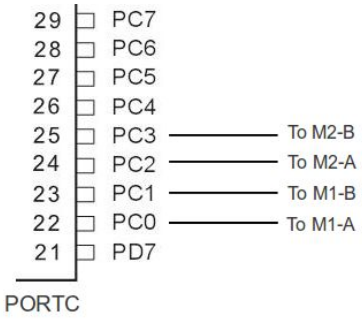


Рисунок 10 – на вхід L293D

Код програми на мові Сі:

```
#include <avr/io.h>
#include <util/delay.h> // for _delay_ms()

int main(void)
{
     DDRC = 0x0F;    // порт С на вихід
                     // мотор під'єднаний через PC0...PC3
     while(1)
     {
     // обертання за годинниковою стрілкою
     PORTC = 0b00000101; // PC0 = High = Vcc
                         // PC1 = Low = 0
                         // PC2 = High = Vcc
                         // PC3 = Low = 0

      _delay_ms(500);     // затримка 0.5 сек

     // обертання проти годинникової стрілки
     PORTC = 0b00001010; // PC0 = Low = 0
                          // PC1 = High = Vcc
                          // PC2 = Low = 0
                          // PC3 = High = Vcc

     _delay_ms(500);       // затримка 0.5 сек}
}
```

Приклад 2. Керування швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 32A.

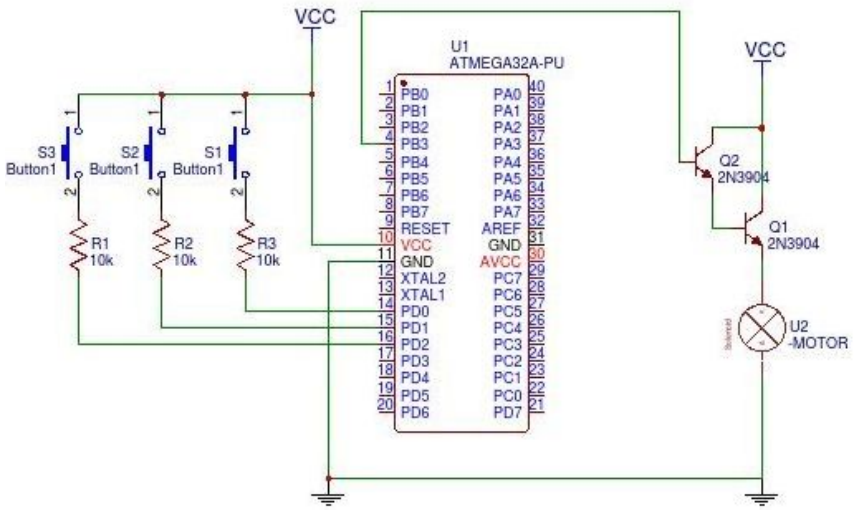


Рисунок 11 – Використання ШІМ сигналів МК для керування швидкістю обертання мотора

Код на мові Сі:

```
#include <avr/io.h>
#define get_bit(reg,bitnum) ((reg & (1<<bitnum))>>bitnum)

int main(void)
{
   // встановити перші 3 штифти порту PORTD на вхід для читання натиску кнопок
   DDRD=0b11111000;
    //  є переконатися що штифт 3 в порті B є на вихід, так як це є штифт OC0
    // який буде продукувати PWM.
```

```
    DDRB=0b11111111;
    PORTD=0b00000000; // ініціалізація порту PORTD нулями
    TCCR0=0b01110101; //конфігурувати TCCR0
    TIMSK=0b00000000;
    // встановити OCR0 в 255 так щоб тривалість циклу спочатку було 0 і мотор
    // не обертався
    OCR0=255;
    while(1)
    {
       if (get_bit(PIND,0)==1)
       {
          OCR0=178; //якщо кнопка 1 натиснута, задати OCR0=178 (duty cycle=10%).
       }
       if (get_bit(PIND,1)==1)
       {
          OCR0=102; //якщо кнопка 2 натиснута, задати OCR0=102 (duty cycle=60%).
       }
       if (get_bit(PIND,2)==1)
       {
           OCR0=25; //якщо кнопка 3 натиснута, задати OCR0=25 (duty cycle=90%).
       }
    }
}
```

Приклад 3. Керування напрямом і швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 32 і моста L293D.

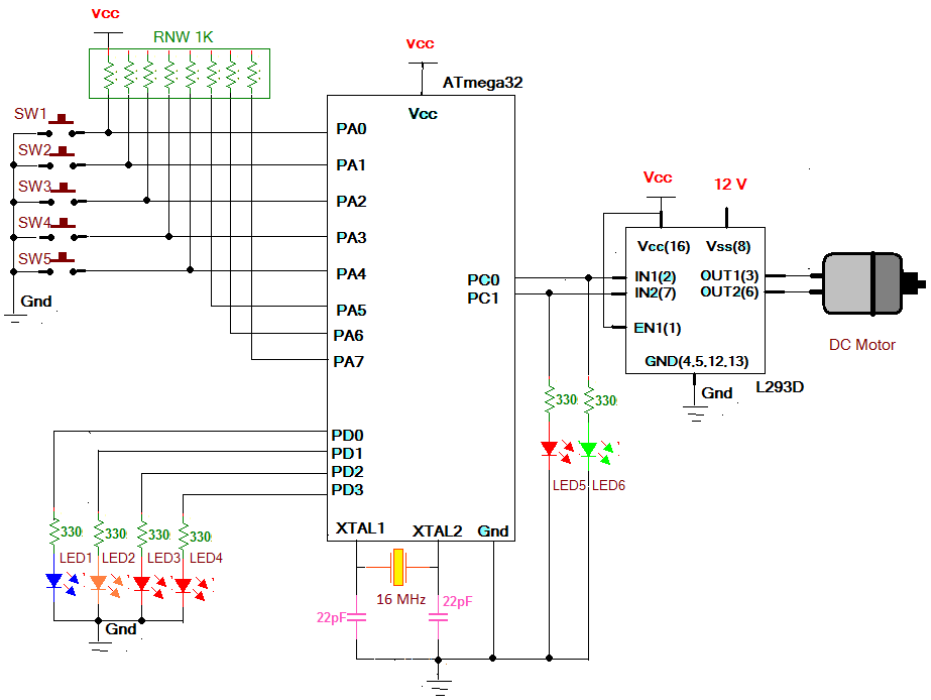


Рисунок 12 – Використання ШІМ сигналів МК і моста IC L293D для керування швидкістю і напрямом обертання мотора

Код на мові Сі:

```
#include <avr/io.h>
#include <util/delay.h>
uint16_t ton=5000,toff=5000;

void keydly()              // затримка усунення брязкоту клавіш
{
```

```
   uint8_t i;
   for(i=0;i<7;i++)
  _delay_loop_2(10000);    // функція бібліотеки циклу затримки
}

void incspeed()            // збільшити ширину імпульсу
{
  PORTD = (0<<PD3);        // led індикація
  if(toff>1000)            // перевірити мінімальний час вимкнення
  {
      toff=toff-500;       // якщо ні, то зменшити час вимкнення
    ton=ton+500;           // і збільшити час включення
  }
  if(toff==1000) PORTD = (1<<PD2);
// якшо мін ліміт досягнуто — включити LED
}

void decspeed()           // зменшити ширину імпульсу
{
  PORTD = (0<<PD2);       // led індикація
  if(ton>1000)            // перевірити мінімальний час включення
  {
     toff=toff+500;       // якшо ні то збільшити час виключення
     ton=ton-500;         // і зменшити час включення
  }
  if(ton==1000) PORTD = (1<<PD3);
                            // якщо мін ліміт досягнуто – виключити LED}

void clockwise()           // обертати мотор за годинниковою стрілкою
{
    while(PINA==0xFF)      // поки люба клавіша не натиснута
    {
       PORTC=0x01;         // застосувати PWM до мотору
       _delay_loop_2(ton);
       PORTC=0x00;
       _delay_loop_2(toff);
    }
}

void anticlockwise()     // обертати мотор проти годинникової стрілки
{
    while(PINA==0xFF)    // поки люба клавіша не натиснута
    {
       PORTC=0x02;       // застосувати PWM до мотору
       _delay_loop_2(ton);
       PORTC=0x00;
       _delay_loop_2(toff);
    }
}

int main(void)
{
    uint8_t r=0,d;       // прапор виконання і напрямку
    DDRC=0x03;           // ініціалізувати порт на вхід і вихід
    DDRD=0x0F;
    DDRA=0x00;
    PORTC=0x00;
    PORTD=0x00;
    while(PINA==0xFF);  //очікувати поки не натиснути люба клавіша
loop: switch(PINA)
           {
           case 0xFE:      // для 1-ої клавіші
                keydly();
                r=1;         // встановити прапор виконання
```

```
                d=0;          // чистка прапора напрямку для CLK напрямку
                clockwise(); // почати обертати мотор за годинниковою стрілкою
                break;
           case 0xFD:         // для 2-оï клавіші
                keydly();
                r=1;         // встановити прапор виконання
                d=1;         // встановити прапор напрямку для ACLK напрямку
                anticlockwise(); // почати обертати мотор проти годин. стрілки
                break;
           case 0xFB:            // для 3-оï клавіші
                PORTC = 0x00;    // зупинити мотор
                PORTD = 0x00;    // всі індикація виключити
                r = 0;           // чистка прапору виконання
                break;
           case 0xF7:            // для 4-оï клавіші
                PORTD=(1<<PD0);  // блимати LED1
                keydly();
                PORTD=(0<<PD0);
                incspeed();      // збільшити швидкість
                if(r==1)         // якщо мотор обертається
                {
                  if(d==0) clockwise();  // обертати за годинниковою стрілкою
                  else anticlockwise();  // або проти годинникової стрілки
                }
                break;
           case 0xEF:            // для 5-оï клавіші
                PORTD=(1<<PD1);  // блимати LED2
                keydly();
                PORTD=(1<<PD1);
                decspeed();     // зменшити швидкість
                if(r==1)        // якшо мотор обертається
                {
                  if(d==0) clockwise();  // підтримувати його обкртання
                  else anticlockwise();
                }
               break;
           }
    goto loop;
}
```

Приклад 4. Керування напрямом і швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 16 і моста L293D.

Код на мові Сі:

```
#define F_CPU 8000000UL             /* визначити частоту ЦП 8 МГц */
#include <avr/io.h>                 /* підключити AVR std. бібліотечний файл*/
#include <avr/interrupt.h>
#include <stdio.h>                  /* підключити std. Бібліотечний файл */
#include <util/delay.h>             /* підключити Delay заголовковий файл */

volatile uint8_t Direction = 0;

void ADC_Init()                    /* функція ініціалізації ADC */
{
   DDRA = 0x00;                /* зробити ADC порт вхідним */
   ADCSRA = 0x87;              /* дозвіл ADC, з freq/128 */
   ADMUX = 0x40;               /* Vref: Avcc, ADC канал: 0 */
}

int ADC_Read(char channel)         /* функція читання ADC */
{
   ADMUX = 0x40 | (channel & 0x07);/* встановити вхідний канал на read */
```

```
   ADCSRA |= (1<<ADSC);              /* Початок ADC перетворень */
   while (!(ADCSRA & (1<<ADIF)));    /* Чекати до кінця перетворень */
   ADCSRA |= (1<<ADIF);              /* Чистка прапору переривання */
   _delay_ms(1);                     /* Невелика затримка */
   return ADCW;                      /* Повернення ADC слова */
}

ISR(INT0_vect)
{
   Direction = ~Direction;           /* Перемкнути напрямок */
   _delay_ms(50);          /* затримка керування програмного усунення брязкоту */
}

int main(void)
{
   DDRC = 0xFF;                      /* встановити порт PORTC на вихід */
   DDRD &= ~(1<<PD2);                /* встановити штифт INT0 на вхід */
   DDRB |= (1<<PB3);                 /* встановити OC0 штифт на вихід */
   GICR = (1<<INT0);                 /* дозволити INT0*/
   MCUCR = ((1<<ISC00)|(1<<ISC01));/* спрацювання INT0 по наростаючому фронту*/
   sei();                            /* дозволити глобальне переривання */
   ADC_Init();                       /* ініціалізувати ADC */
   TCNT0 = 0;                        /* встановити лічильник таймера timer0 в 0 */
   /* задати Fast PWM з Fosc/64 Timer0 clock */
   TCCR0 = (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS00)|(1<<CS01);
   while(1)
   {
     if (Direction !=0)  /* обертати DC мотор за годинниковою стрілкою */
        PORTC = 1;
     else              /* інакше обертати DC мотор проти годинникової стрілки */
        PORTC = 2;
     /* читати ADC і відображувати в 0-255 для запису в OCR0 регістр */
     OCR0 = (ADC_Read(0)/4);
   }
}
```

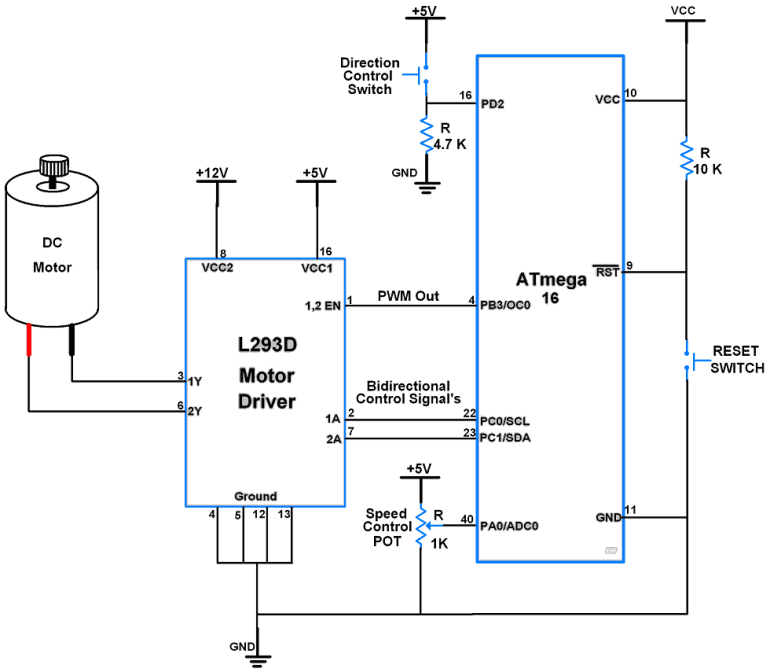


Рисунок 13 – Використання ШІМ сигналів МК і моста IC L293D для керування швидкістю і напрямом обертання мотора

Завдання.

1. Промоделювати в Proteus схему рис. 8 для отримання ШІМ сигналів з коефіцієнт заповнення 0.2.

2. Промоделювати в Proteus схему рис. 8 для отримання ШІМ сигналів з коефіцієнт заповнення 0.5.

3. Промоделювати в Proteus схему рис. 8 для отримання ШІМ сигналів з коефіцієнт заповнення 0.8.

4. Промоделювати в Proteus схему рис. 9 для керування двигуном постійного струму.

5. Промоделювати в Proteus схему рис. 10 (приклад 1) для керування напрямом обертання мотора.

6. Промоделювати в Proteus схему рис. 11 (приклад 2) для керування швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 32A.

7. Промоделювати в Proteus схему рис. 12 (приклад 3) для керування напрямом і швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 32 і моста L293D.

8. Промоделювати в Proteus схему рис. 13 (приклад 4) для керування напрямом і швидкістю обертання мотора з використанням ШІМ сигналів МК ATmega 16 і моста L293D.

Питання.

1. Принцип керування швидкістю мотора за допомогою ШІМ.
2. Що таке коефіцієнт заповнення і шпаруватість ШІМ сигналу?
3. Для чого призначений таймер 555. Які його режими роботи?
4. Як змінити напрямок обертання мотора. Для чого призначені Н-мости?
5. Для чого призначена мікросхема L293D?
6. Як генеруються ШІМ сигнали в МК?

СПИСОК ЛІТЕРАТУРИ

1. Новацький А. О. Мікропроцесорні та мікроконтролерні системи. Частина 2. Проектування мікропроцесорних систем. Підручник. / Новацький Анатолій Олександрович. – НТУУ «Київський політехнічний інститут імені Ігоря Сікорського» – К.: Видавництво «Політехніка», 2021. – 462 с.

2. Смірнов В. В. Програмування мікроконтролерних систем : навч. посіб. / В. В. Смірнов, Н. В. Смірнова, Ю. М. Пархоменко. – Кропивницький : Центральноукр. НТУ, 2021. – 262 с.

3. Грищук Ю. С. Мікроконтролери: Архітектура, програмування та застосування в електромеханіці : навч. посіб. / Ю. С. Грищук. – Харків : НТУ «ХПІ», 2019. – 384 с.

4. Програмування мікроконтролерів AVR : [навчальний посібник] / С. М. Цирульник, О. Д. Азаров, Л. В. Крупельницький, Т. І. Трояновська. – Вінниця : ВНТУ, 2018. – 111 с.

5. Програмування мікроконтролерів систем автоматики : конспект лекцій для студентів базового напряму 050201 “Системна інженерія” / Укл.: А. Г. Павельчак, В. В. Самотий, Ю.В. Яцук – Львів : Львівська політехніка. – 2012. – 143 с.

6. Трамперт В. Измерение и регулирование с помощью AVR-микроконтроллеров.: Пер. с нем. – К.: “МК-Пресс”, 2007. – 208 с.

7. Трамперт Вольфганг. AVR-RISC микроконтроллеры. – Пер. с нем. – К.: “МК-Пресс”, 2006. – 464 с.

8. Richard Barnett, Larry O’Cull, Sarah Cox. Embedded C programming and the Atmel AVR. – NY: Delmar, Cengage Learning, 2006. – 532 p.

9. Сайт фірми Microchip: [Електронний ресурс] / Microcontrollers & Microprocessors. – Режим доступу: https://www.microchip.com, вільний. – Загол. з екрану. – Мова анг.