МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДВНЗ «УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

Факультет інформаційних технологій

Кафедра програмного забезпечення систем

«НИЗЬКОРІВНЕВЕ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ»

Методичні вказівки до лабораторних робіт

УЖГОРОД – 2018

Низькорівневе програмування мікроконтролерів: методичні вказівки до лабораторних робіт для студентів за напрямом підготовки 121 «Інженерія програмного забезпечення» факультету інформаційних технологій УжНУ / Розробники: Лях І.М., Поліщук В.В.– Ужгород: 2018. – 51 с.

У методичних вказівках до лабораторних робіт з курсу «Низькорівневе програмування мікроконтролерів» розглянуто п’ять лабораторних робіт, що входять до складу робочої програми. Наведено теоретичний матеріал необхідний для виконання лабораторної роботи. До лабораторних робіт сформульовано завдання студентам, вимоги до порядку виконання та змісту звіту по проробленій роботі. У методичних вказівках наведений перелік запитань на підсумковий контроль.

Укладачі:

- к.т.н. Лях І.М., доцент кафедри інформатики та фізико-математичних дисциплін факультету інформаційних технологій ДВНЗ «УжНУ»;
- к.т.н. Поліщук В.В., доцент кафедри програмного забезпечення систем факультету інформаційних технологій ДВНЗ «УжНУ».

Рецензенти:

- к.ф-м.н., доц., завідувач кафедри програмного забезпечення систем ДВНЗ «УжНУ» Білак Ю.Ю.
- к.т.н., доцент кафедри інформатики та фізико-математичних дисциплін факультету інформаційних технологій ДВНЗ «УжНУ» Кляп М. М.

Рекомендовано кафедрою програмного забезпечення систем від «31» серпня 2018 р., протокол №1.

ЗМІСТ

Вступ

Метою дисципліни «Низькорівневе програмування мікроконтролерів» є вивчення принципів і методів розробки програмного забезпечення для мікроконтролерів, ознайомлення із програмними та апаратними засобами розробки, відлагодження і програмування сучасних мікроконтролерів, що широко використовуються у вимірювальній і обчислювальній техніці, в мікропроцесорних та програмних засобах автоматизації.

Завдання дисципліни:

- ознайомлення з галузями використання, класифікацією та можливостями сучасних мікроконтролерів, апаратними та програмними засобами для програмування мікроконтролерів;
- формування уявлень про принципи та типові алгоритми роботи пристроїв на базі мікроконтролерів;
- вивчення типових схем підключення та прийомів програмування мікроконтролерів, для обміну даними з іншими пристроями;
- формування навичок розробки програм для мікроконтролерів.

В результаті вивчення даної дисципліни студент повинен:

знати: принципи розробки програмного забезпечення для мікроконтролерів, сучасну базу мікроконтролерів та засобів для роботи з ними;

вміти: самостійно обирати засоби мікропроцесорної техніки для реалізації конкретних пристроїв, вибирати програмні та апаратні засоби для роботи з ними, проектувати, розробляти та підлагоджувати програми для мікроконтролерів.

Лабораторна робота №1. Програмне середовище AVR Studio

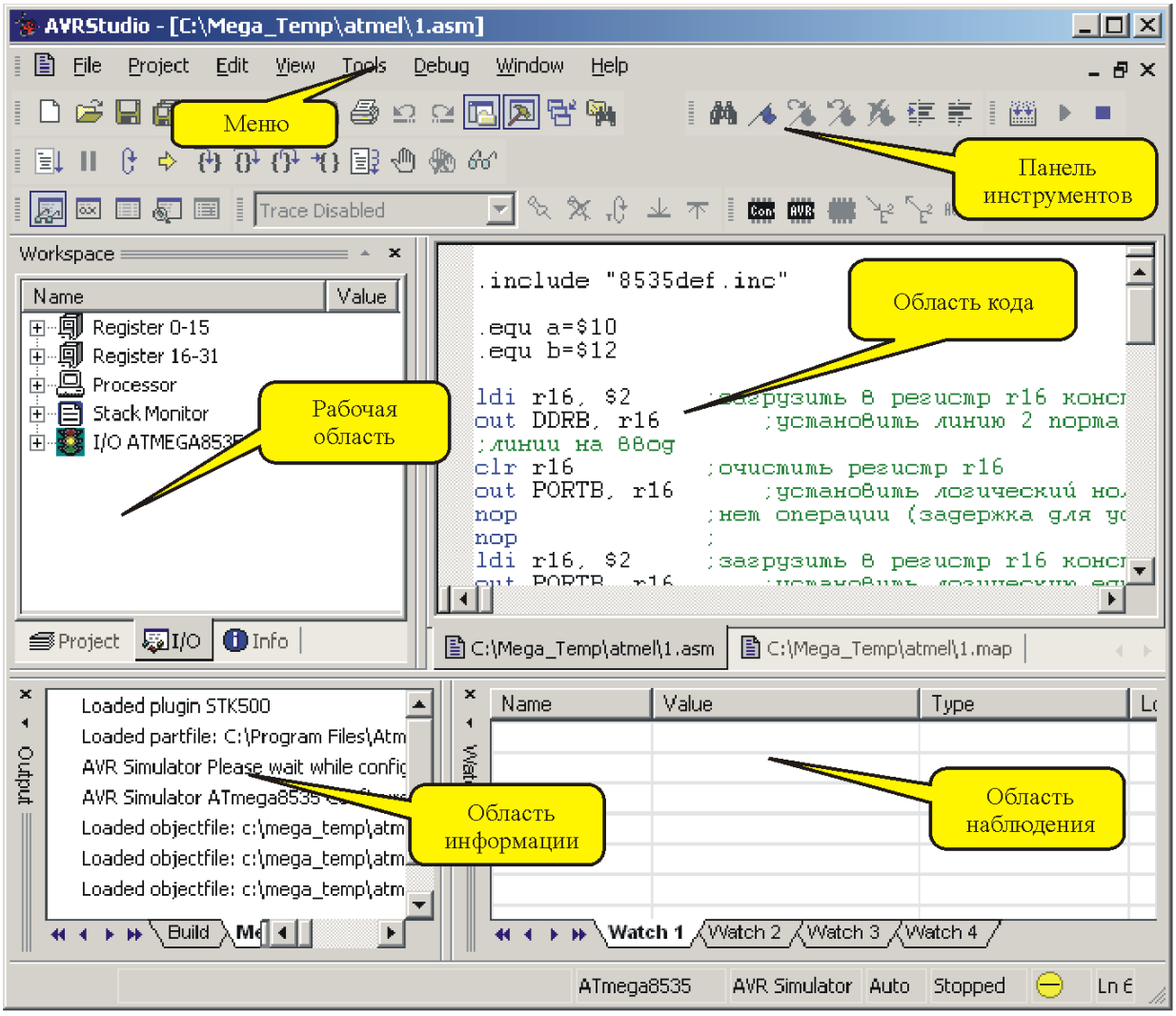
Мета роботи: вивчити основи роботи в програмному середовищі-симуляторі AVR Studio.

Теоретичні відомості

AVR Studio є інтегрованим середовищем розробки програмного забезпечення для мікроконтролерів серії AVR. В даному середовищі можна створювати код на асемблері або при підключенні стороннього С-компілятора, на мови високого рівня С, проводити компілювання коду з використанням інтегрованого симулятора або підключеного внутрішньо-схемного емулятора.

Розглянемо основні вікна та елементи керування середовища AVR Studio.

Основне вікно AVR Studio.



Меню містить команди керування файлами (file), керування проектом (Project), редагування (Edut), керування виглядом (View), інструменти (Tools), компілювання (Debug), керування вікнами (Window), довідка (Help).

Панель інструментів містить найбільш часто використовувані кнопки команд.

Робоча область містить три вкладки: Project - управління файлми проекту; I / O - вкладка перегляду і редагування регістрів; Info - вкладка інформації про мікроконтролер.

В області коду проводитися перегляд і редагування тексту програми, спостереження за ходом виконання програми.

В області інформації містяться вкладки: Build - інформація асемблера; Messages - вікно повідомлень; Find in files - вікно пошуку.

У вікні спостережень можна переглядати стан змінних під час виконання програми.

У вкладці I / O міститься п'ять груп для перегляду вмісту регістрів 0-15, регістрів 16-31, стану процесора, стека і регістрів введення-виведення внутрішніх пристроїв мікроконтролера.

Група процесора містить інформацію про ядро мікроконтролера.

У цій групі можна переглянути наступну інформацію:

- **Program Counter** (Програмний лічильник) - вказує адресу наступної команди, яка буде виконана. Адреса команди відображається в шістнадцятковому вигляді, і може бути змінений, коли виконання зупинено.

- **Stack Pointer** (Покажчик стека) - містить поточне значення вказівника стека. Значення показника стека може бути змінено, коли виконання зупинено.

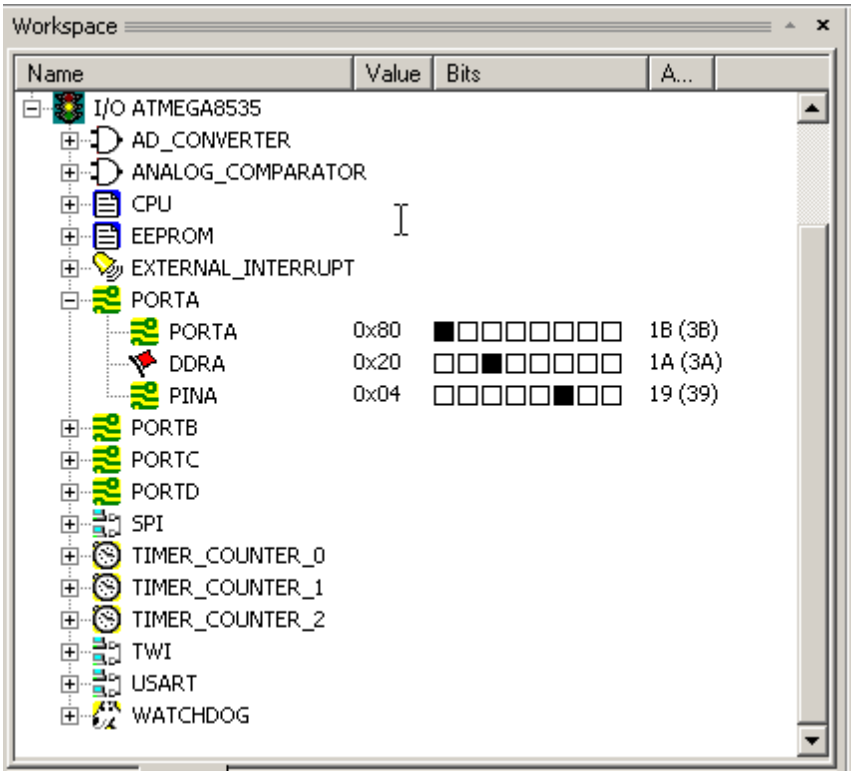
- **Cycle Counter** (Лічильник циклів) - лічильник циклів відображає число тактових циклів, які пройшли починаючи з останнього скидання. Значення Лічильника циклів відображено як десяткове значення і може бути змінено, коли виконання зупинено.

- **X-register, Y-register, Z-register** - регістри-показники X, Y і Z.

- **Frequency** (Частота) - частота роботи мікроконтролера.

- **Stop Watch** - час, що минув з початку виконання програми.

Група введення-виведення внутрішніх пристроїв мікроконтролера призначена для перегляду стану всіх регістрів внутрішніх пристроїв.



Вікно введення-виведення містить чотири стовпці. Перший стовпець відображає назву регістра або групи регістрів.

Другий стовпець відображає значення регістра, в якому дані можуть бути відображені і відредаговані різними способами.

Третій стовпець відображає вміст регістра в бітовій формі.

Четвертий стовпець відображає адресу регістра в шістнадцятковій формі.

Назви регістрів, значення яких змінилися на останньому кроці виконання програми, відображаються червоними символами.

Елементи керування

Інтерфейс програмного середовища AVR Studio побудований з урахуванням загально-прийнятих принципів стандартного інтерфейсу 32-розрядних версій Windows, тому значки кнопок стандартних операцій, таких як створення нового файлу, відкриття файлу, збереження поточного документа, збереження всіх відкритих документів, копіювання, вставка, скасування останньої операції, крок вперед по списку зроблених відмін, пошук нічим не відрізняються від однойменних в інших додатках Windows.

Більш детально зупинимося на специфічних командах програмного середовища AVR Studio.

ввімкнути / вимкнути відображення робочої області.

 ввімкнути / вимкнути відображення області інформації.

 встановити закладку.

 перейти до наступної закладки.

 перейти до попередньої закладці.

 видалити всі закладки.

 скомпілювати.

 запустити налагодження. Компілює поточний проект і, за умови відсутності помилок компіляції, запускає програму на виконання.

 зупинити налагодження.

 запуск програми на виконання.

 призупинити виконання програми.

 імітує апаратне скидання мікроконтролера.

 виконує крок програми з входом в викликані процедури (підпрограми).

 виконує крок програми без входу в викликані процедури (підпрограми). Виконання викликаної процедури (підпрограми) відбувається за один крок.

 виконує програму до виходу з поточної процедури (підпрограми).

 виконує програму до рядка, в якій розташований курсор.

 запускає програму на виконання.

 встановлює точку зупину для налагодження програми.

 очищає всі точки зупинки.

 вмикає і вимикає відображення вікна спостереження.

 вмикає і вимикає відображення вікна регістрів.

 вмикає і вимикає відображення вікна пам'яті.

Порядок виконання роботи

1. Запустіть AVR Studio. У діалоговому вікні натисніть кнопку Create New Project.

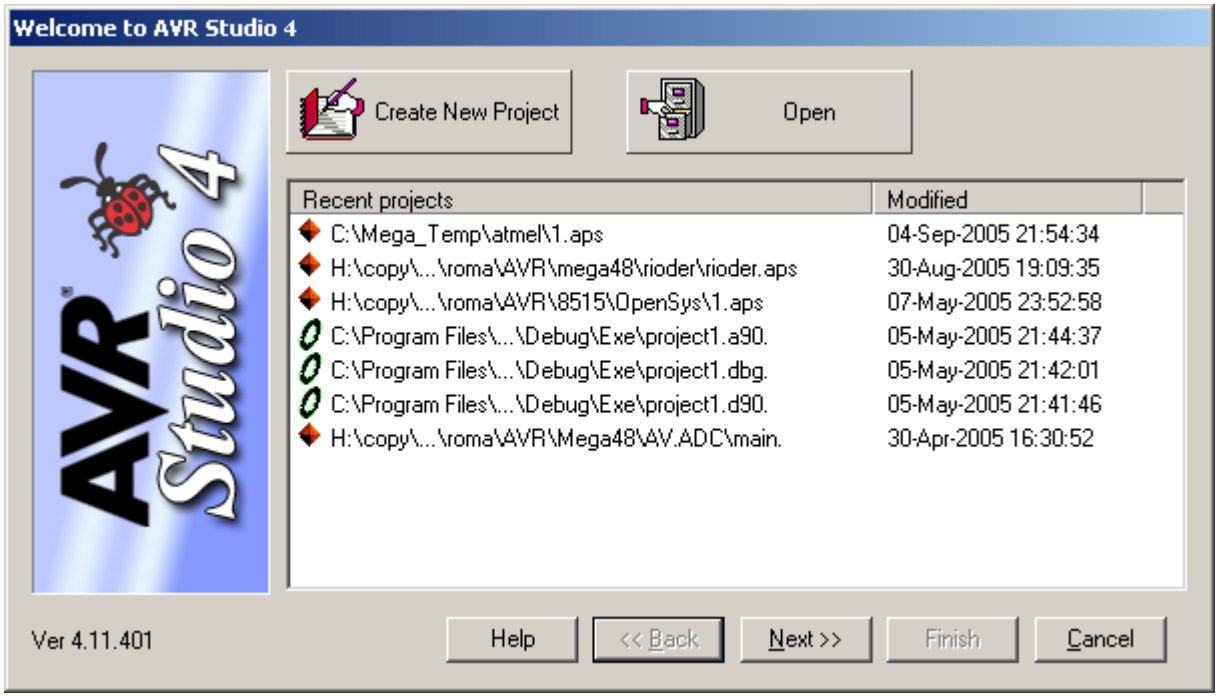
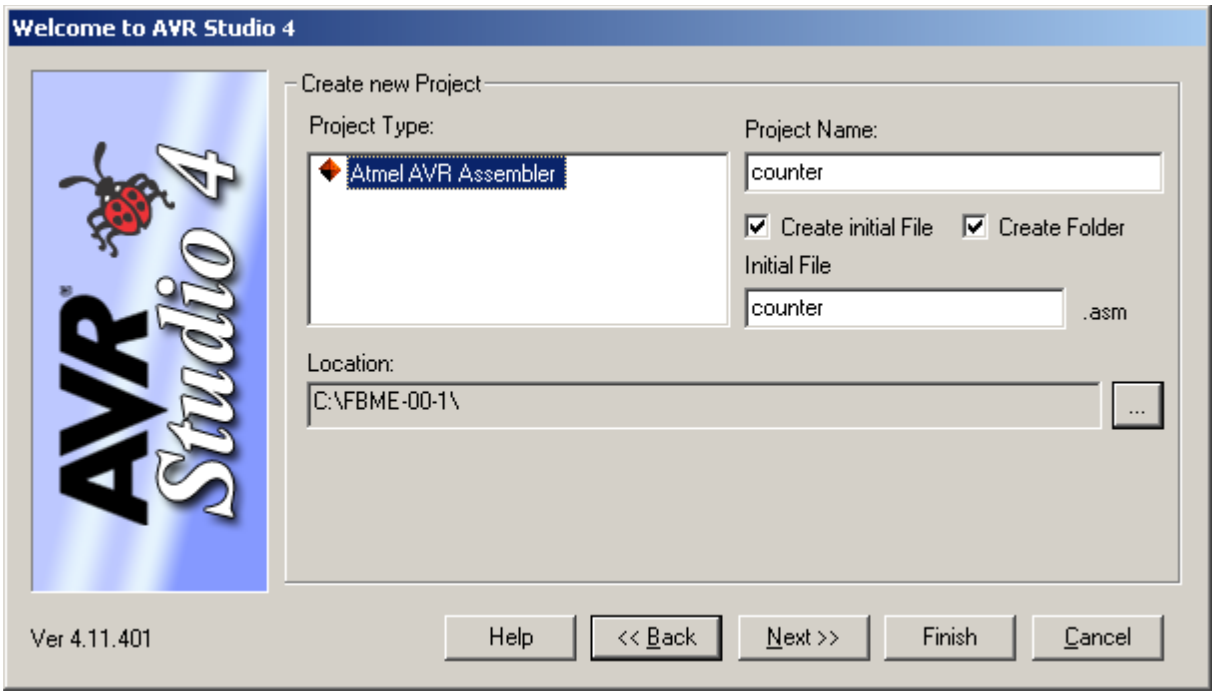


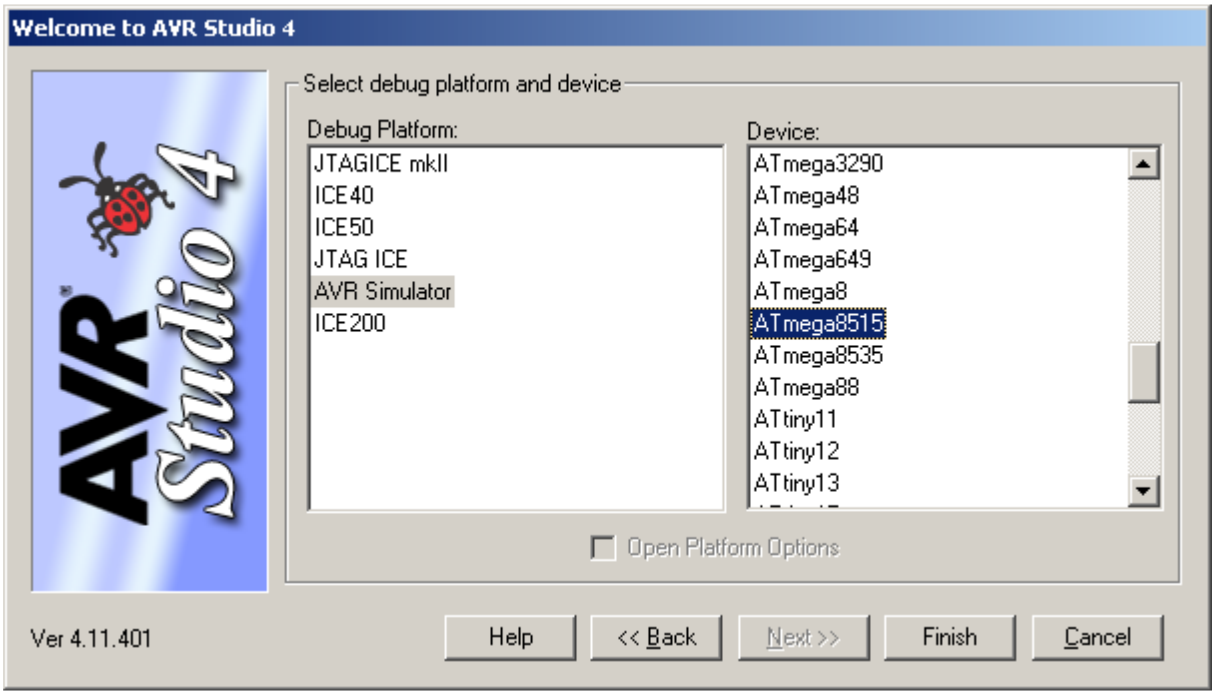
Рис. 1.3. Створення проекту

2. У наступному діалоговому вікні виберіть тип проекту (Project Type)

- Atmel AVR Assembler, введіть ім'я проекту "counter" (Project Name), підтвердіть необхідність створення початкового файлу і папки (Create initial File, Create Folder), виберіть створену папку в якості розташування для проекту (Location) і натисніть кнопку "Далі" (Next).



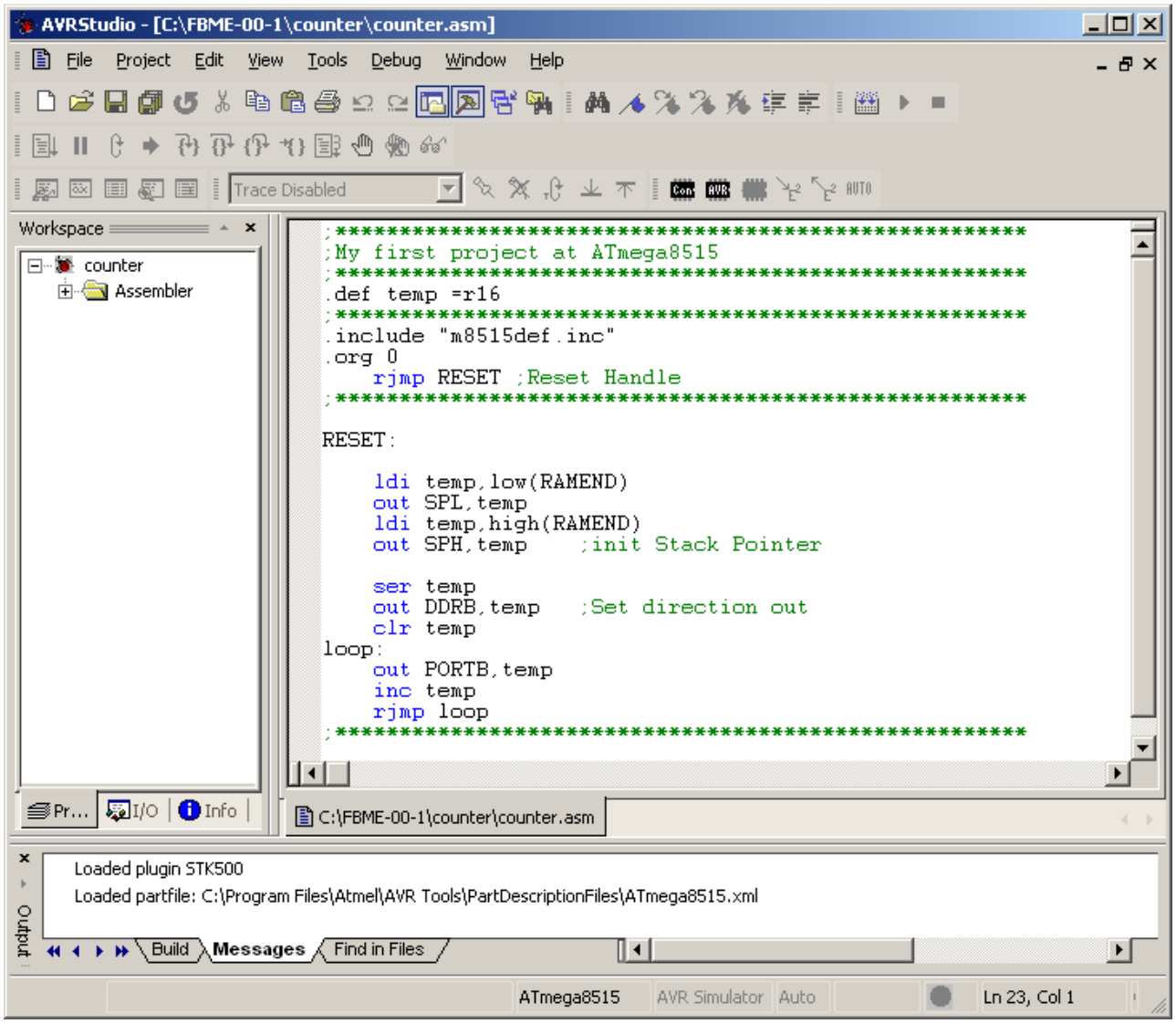
3. Виберіть в якості платформи для відладки (Debug Platform) AVR Simulator (симулятор) і мікроконтролер (Device) ATmega8515. Натисніть кнопку "Завершити" (Finish).



4. У вікні вбудованого редактора введіть наступний код:

```
;*****************************************************
;My first project at ATmega8515
;*****************************************************
.def temp =r16
;*****************************************************
.include "m8515def.inc"
.org 0
rjmp RESET ;Reset Handle
;*****************************************************
RESET:
ldi temp,low(RAMEND)
out SPL,temp
ldi temp,high(RAMEND)
out SPH,temp ;init Stack Pointer
ser temp
out DDRB,temp ;Set direction out
clr temp
loop:
out PORTB,temp
inc temp
rjmp loop
;*****************************************************
```

Після вводу коду редактор повинен виглядати наступним чином:



5. Для проведення компіляції коду виберіть команду Build з меню Project або натисніть клавішу F7. Результат компіляції відображається в області інформації. У нашому проекті видно, що програма займає 22 байта і що компіляція завершена без помилок і попереджень.

6. В результаті компіляції проекту середовищем AVR Studio був створений файл, необхідний для проведення програмної симуляції роботи мікроконтролера. Для початку відладки виберіть команду Start Debugging з меню Debug, або натисніть кнопку .

Зверніть увагу на що з'явився у вікні редактора коду покажчик команди. Він має вигляд жовтої стрілки, розташованої в лівій стороні вікна редактора. Вказівник розташований навпроти команди, яка виконається на наступному кроці компілювання.

7. Для того щоб візуально спостерігати за роботою програми потрібно скористатися вкладкою введення-виведення робочої області. У вікні вводу-виводу натисніть значок "+" напроти групи регістрів порту B (PORTB). В результаті у вікні з'являться три регістра порту B: PORTB (реєстр даних порту

B), DDRB (реєстр напрямки порту B), PINB (реєстр даних на зовнішній вивід порту B).

8. Натискаючи клавішу F11 або кнопку «Step into»  простежте за виконанням коду і його впливом на стан регістрів порту B.

Розглянемо докладніше код програми.

Рядок, розташована після знака ";" є коментарем і при компіляції опускається.

```
.def temp =r16
```

Дана директива привласнює символічну назву "temp" регістру загального призначення r16.

```
.include "m8515def.inc"
```

Дана директива підключає до компіляції файл опису апаратно-залежних імен регістрів, бітів, векторів переривань і інших параметрів мікроконтролера ATmega8515.

```
.org 0
```

Дана директива повідомляє асемблеру про те, що наступна за даною директивою команда повинна бути розташована в пам'яті програм за адресою 0x00. Адреса 0x00 є адресою вектора скидання.

```
rjmp RESET
```

Команда rjmp здійснює відносний безумовний перехід програми до зазначеної мітці або адресою.

```
RESET:
```

Для позначення мітки використовується знак ":" в кінці назви. Мітки використовують для можливості реалізації переходів з вказівкою символьних імен.

```
ldi temp,low(RAMEND)
```

Дана команда завантажує в регістр "temp" молодший байт константи "RAMEND". RAMEND має значення кінцевої адреси оперативної пам’яті мікроконтролера. Сама ж константа RAMEND задається директивою equb файлі опису мікроконтролера "m8515def.inc".

```
out SPL,temp
```

Дана команда завантажує в регістр покажчика стека (молодший байт) SPL вміст регістра "temp".

```
ldi temp,high(RAMEND)
```

Дана команда завантажує в регістр "temp" старший байт константи "RAMEND".

```
out SPH,temp
```

Дана команда завантажує в регістр покажчика стека (старший байт) SPH вміст регістра "temp".

```
ser temp
```

Дана команда встановлює регістр "temp" (завантажує в нього значення 255).

```
out DDRB,temp
```

Дана команда завантажує в регістр напрямок порту B (DDRB) вміст регістра "temp". Установка бітів в DDRB налаштовує порт B на вивід.

```
clr temp
```

Дана команда очищає регістр "temp" (завантажує в нього значення0).

```
loop:
```

Мітка "loop" використовується для створення основного циклу програми.

```
out PORTB,temp
```

Дана команда завантажує в регістр PORTB вміст регістра "Temp".

```
inc temp
```

Дана команда інкримінує вміст регістра "temp".

```
rjmp loop
```

Дана команда здійснює відносний безумовний перехід програми до мітки "loop", замикаючи основний цикл програми.

Контрольні питання

1. Опишіть призначення вікон програмного симулятора AVR Studio.
2. Що таке програмний лічильник?
3. Для чого використовується покажчик стека?
4. Які типи пам'яті присутні в мікроконтролерах серії AVR?
5. Опишіть команди Trace into, Step over, Step out і назвіть їх відмінності.
6. Для чого використовуються мітки?

Необхідне програмне забезпечення та обладнання

1. AVR Studio v.4.11
2. Повний опис набору інструкцій мікроконтролерів серії AVR "Instruction Set.pdf"

Лабораторна робота №2. Структурна схема навчально-налагоджувального стенду

Мета роботи: розглянути структуру стенда EV8031 / AVR, вивчити принципироботи зі стендом EV8031 / AVR

Теоретичні відомості

Стенд EV8031 / AVR являє собою мікроконтроллерні системи, засновану на мікроконтролері ATmega8515 і оснащену зовнішнімОЗУ і різноманітними периферійними пристроями. Він дозволяє компілювати програми, написані на мовах Сі та Асемблер.

Завантаження програми здійснюється з персонального комп'ютера з допомогою програматора, підключеного до паралельного порту LPT.

Структурна схема навчально-налагоджувального стенда EV8031 / AVR представлена на малюнку 2.1.

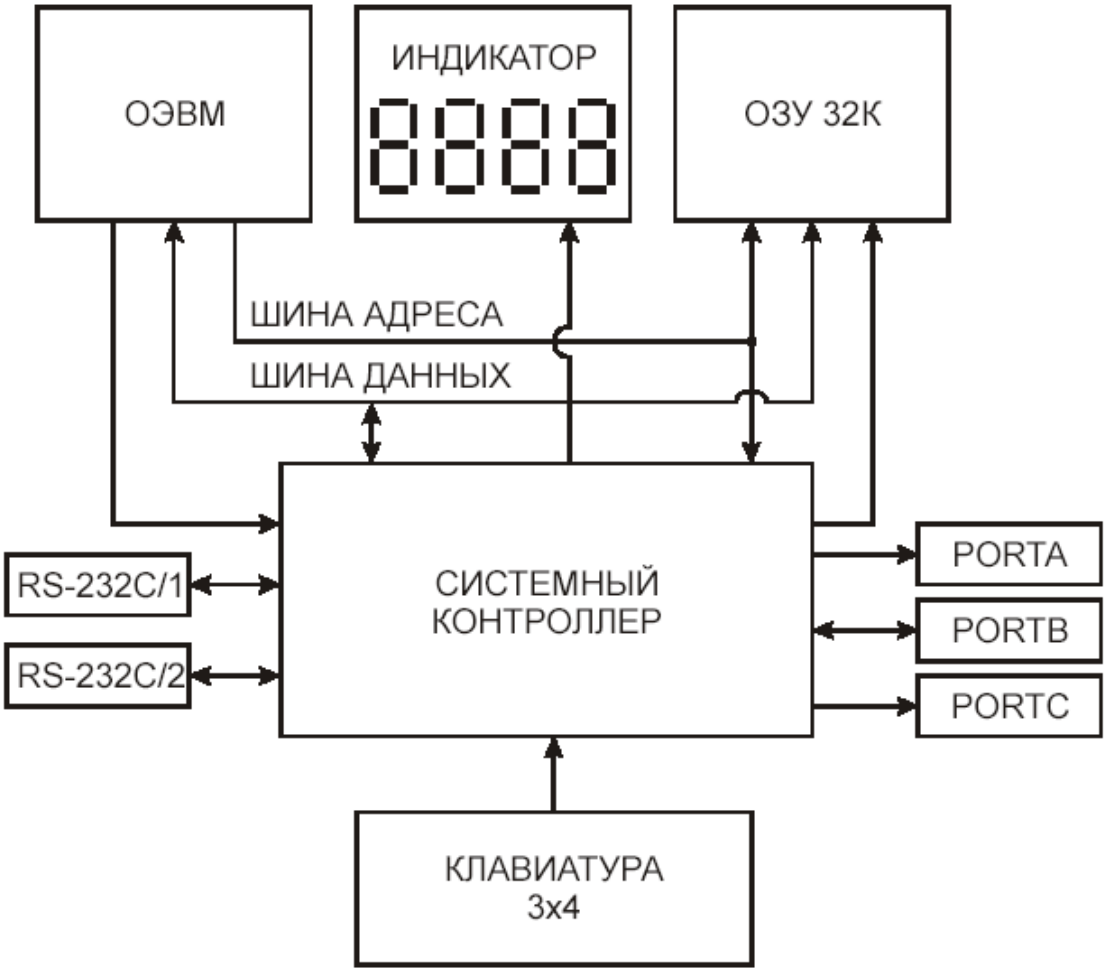


Рис. 2.1 - Структурна схема стенду EV8031 / AVR

Схема розташування елементів на стенді EV8031 / AVR, а також на платі розширення представлена на малюнку 2.2.

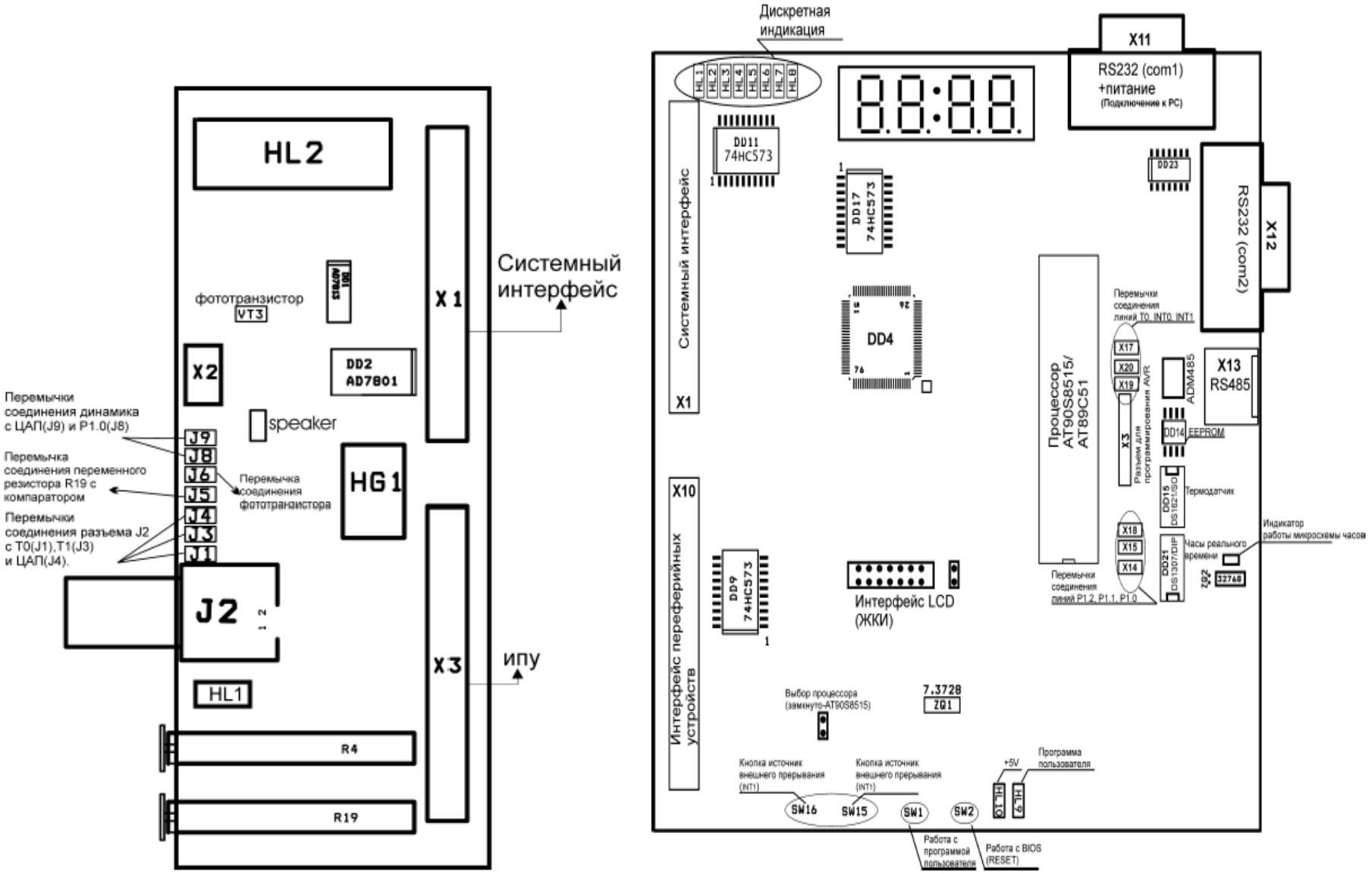


Рис. 2.2 - Схема розташування елементів стенду і плати розширення:

HG1 – знакосинтезуючий індикатор 5x7, HL2 - 4-х розрядна динамічна індикація, HL1 – світлодіодний індикатор спрацьовування компаратора, R4 - змінний резистор, що змінює частоту генератора, R19 – змінний резистор, джерело вхідного сигналу АЦП

Адресація (звернення) мікроконтролера до периферійних пристроїв стенду реалізовано як адресація до осередків пам'яті в адресному просторі від 8000H до FFFFH. Сигнали вибірки периферійних пристроївформуються дешифратором адреси всередині мікросхеми системного контролера DD4.

Периферійні пристрої стенду EV8031 / AVR і плати розширення

Послідовний приймально-передавач. Модуль послідовного зв'язку сформований на мікросхемі приймача 1489, передавача 74HC04,мультиплексора каналу передачі (всередині системного контролера). Швидкістьобміну по послідовному порту в програмі яка компілюється може бутивстановлена записом конфігураційних бітів у відповідні регістримікроконтролера ATmega8515.

Вибір каналу послідовної передачі здійснюється сигналами CFG0, CFG1 за адресою 9001H. Встановлення цих бітів в "логічний нуль" включає порт 1, на схемі X11, цей порт має неповний набір сигналів (RxD, TxD, RI) і призначений для обміну даними з ПК.

Програмна установка сигналів CFG0 в "0", а CFG1 в "1" формує вибірку додаткового каналу послідовної передачі даних, роз’єм X12. Додатковий послідовний канал має повний набір сигналів інтерфейсу RS-232C.

Світлодіодний індикатор. Чотирирозрядний семисегментний світодіодний індикатор підключений до системного контролеру, який автоматично виконує динамічну регенерацію і декодування двійкового коду в код семи сегментного індикатора. Індикатор працює завжди, відразу після подачі живлення. Контролер індикатора містить два восьмирозрядних регістра, вміст яких відображається на індикаторі. Вміст регістра з адресою 0xA000 відображається на двух лівих розрядах, вміст регістра з адресою 0xA001 (0xB000) - на двох правих розрядах в шістнадцятковій формі. Керування десятковими точками і гасінням здійснюється через регістр DC_REG (0xA004). Біти DP3 ... DP0 керують десятковими точками. Запис "1" в відповідний розряд включає десяткову точку. Біти BL3 ... BL0 керують гасінням розрядів індикатора. Запис "1" в ці біти викликає гасіння відповідного розряду індикатора.

Матрична клавіатура. Стан стовпця матриці клавіатури зчитується з осередку з базовою адресою 0x9000, біти 3 ... 0. Відповідний стовпець вибирається нулем в розрядах адреси A2 ... A0. Тобто, адреса 0x9006 вибирає перший стовпець, адреса 0x9005 - другий стовпець, адреса 0x9004 - третій стовпець. Ознака натиснутої кнопки зчитується як нуль в відповідному розряді.

Цифроаналоговий перетворювач. ЦАП виконаний на мікросхеміAD7801 DD2 (8-розрядний ЦАП). Вхідними сигналами для ЦАП єлінії AD7-AD0. Вихідний сигнал знімається з роз'єму BNC.

Аналого-цифровий перетворювач. АЦП виконаний на мікросхемі ЦАП AD7801, операційному підсилювачі, який використовується в якості компаратора LM358 DA1. Вхідним аналоговим сигналом для АЦП є сигнал з змінного резистора R19. Лінії AD7-AD0 використовуються для формування цифрового вхідного коду. На виході ЦАП формується напруга, пропорційна вхідному коду. Сигнал спрацювання компаратора знімається з (DA1-2) входу мікроконтролера P1.7. Спрацьовування компаратора візуально видно по загоряння світлодіода HL1. Якщо на P1.7 "0" – світлодіод світиться.

Генератори. У схемі присутній генератор із змінною частотою генерації ~ 1 - 50 кГц, елементи R1, R4, R5, R7, R10, R11, R15, R16, C3, VT1, DA1-1 (зміна частоти здійснюється за допомогою резистора R4), і генератор з фіксованою частотою ~ 10кГц, елементи R19, R20, C16, DD18-1, DD18-2, DD18-3.

Динамічна індикація. Являє собою 4-розрядний семисегментний індикатор HL2 ввімкнений за схемою динамічної індикації.

Управління динамічною індикацією здійснюється за допомогою елементів DD3 (лінія даних A, B, C, D, E, F, G, DP, - PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7) сигнали надходять з порту PB, сигнали вибірки відповідного розряду надходять від ліній PC0, PC1 порту C. Схема включення динамічної індикації представлена на малюнку 2.3.

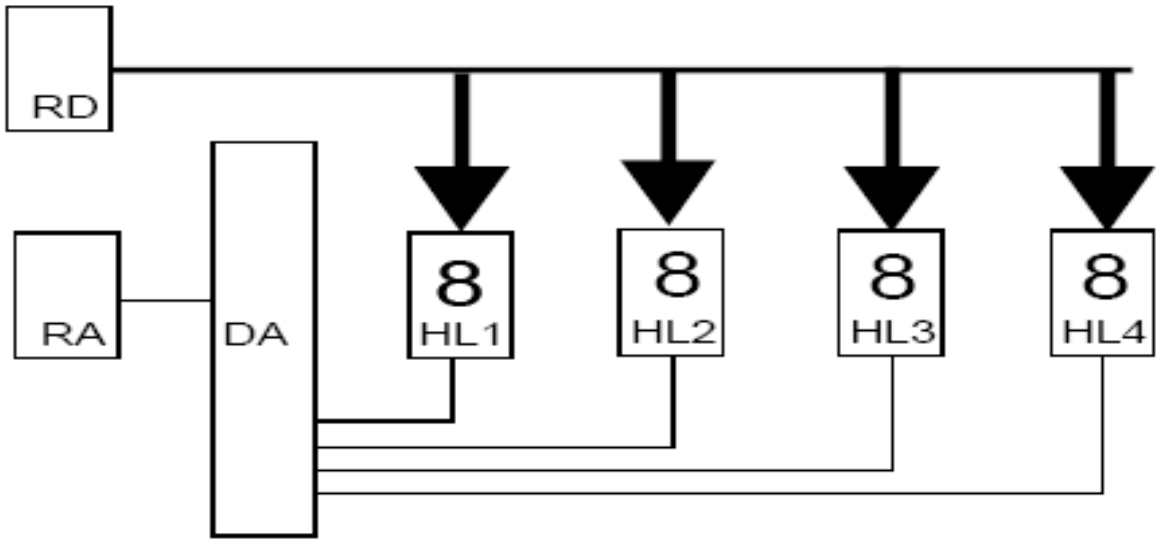


Рис. 2.3. - Схема включення динамічної індикації

Матричний індикатор. Мікроконтролерний стенд EV8031 / AVR оснащений матричним індикатором розміром 5x7. Керування матричним індикатором здійснюється сигналами безпосередньо з портів мікроконтролера ATmega8515.

Для реалізації звернень до безлічі периферійних пристроїв системним контролером здійснюється декодування адресної інформації. Карта портів вводу/виводу стенду EV8031/AVR приведена в таблиці 2.1.

Таблиця 2.1 - Карта портів вводу/виводу стенду EV8031/AVR.

Адрес	Тип цикла	B7	B6	B5	B4	B3	B2	B1	B0	Имя
Порты периферийных устройств										
8xx0	Запись	[Порт A]								PA_REG
8xx1	Запись	[Порт B]								PB_REG
8xx2	Запись	[Порт C]								PC_REG
8xx3	Запись	x	x	x	x	x	TRISC	x	x	TRIS
ЖКИ										
8xx4	Запись	Регистр команд ЖК индикатора								LCD_CMD
8xx5	Запись	Регистр данных ЖК индикатора								LCD_DATA
Последовательный порт										
9xxx	Чтение	CTS	DSR	DCD	RI	KL3	KL2	KL1	KL0	US_REG
Cxx0	Запись	x	x	X	x	DTR	RTS	CFG1	CFG0	UC_REG
Индикатор и светодиоды										
Axx0	Запись	[Регистр индикатора 0]								DISPLAY[0]
Axx1	Запись	[Регистр индикатора 1]								DISPLAY[1]
Axx2	Запись	<зарезервировано>								DISPLAY[2]
Axx3	Запись	<зарезервировано>								DISPLAY[3]
Axx4	Запись	DP3	DP2	DP1	DP0	BL3	BL2	BL1	BL0	DC_REG
Axx5	Запись	<зарезервировано>								EDC_REG
Axx6	Запись	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0	LED_REG
Управление работой										
Axx7	Запись	x	x	X	x	x	x	x	RUN	SYS_CTL
Совместимые регистры										
Bxx0	Запись	[Регистр индикатора 1]								DISPLAYB

Порядок виконання роботи

1. Ознайомитися зі структурою стенду EV8031 /AVR, зі складом периферійних пристроїв.

2. Ознайомитися з картою портів вводу/виводу стенду EV8031/AVR іпринципами роботи з периферійними пристроями.

3. Намалювати структурну схему стенда EV8031/AVR.

4. Описати принципи звернення до периферійних пристроїв стендуEV8031/AVR.

Контрольні питання

1. Поясніть структурну схему стенда EV8031/AVR.
2. Поясніть схему розташування елементів стенду EV8031/AVR.
3. Як здійснюється звернення мікроконтролера до периферійнихпристроїв.
4. Опишіть функціональний склад стенду EV8031/AVR.
5. Поясніть принцип роботи зі світлодіодним індикатором і матричною клавіатурою.
6. Поясніть принцип роботи з динамічною індикацією і матричним індикатором.

Необхідне програмне забезпечення та обладнання

1. Навчально-лабораторний стенд EV8031/AVR
2. Повний опис мікроконтролера ATmega8515 - "Atmega8515.pdf".

Лабораторна робота №3. Основи мови програмування асемблер для мікроконтролерів серії AVR

Мета роботи: вивчити основи асемблера AVRASM для мікроконтролерів серії AVR

Теоретичні відомості

Програмний код асемблера складається з мнемонік команд, міток і директив. Команди директиви мають операнди. Довжина лінії коду не повинна перевищувати 120 символів.

Кожна з ліній коду може починатися міткою, що складається з алфавітно-цифрових символів завершеною двокрапкою. Мітки використовуються для команд переходу і розгалуження, а також як імена змінних і таблиць в пам'яті програм, EEPROM і ОЗУ.

Лінії коду можуть бути наступних 4-х форматів:

```
[label:] directive [operands] [Comment]
[label:] instruction [operands] [Comment]
Comment
Empty line
```

Коментарі мають наступний формат:

```
; [Text]
```

Елементи поміщені в квадратні дужки "[", "]" необов'язкові.

Приклад:

```
label:      .EQU var1=100 ; встановити змінну var1 в 100 (Directive)
      .EQU var2=200     ; встановити змінну var2 в 200
test:  rjmp test        ; нескінченний цикл (команда)
                        ; чиста лінія коментаря
```

Врахуйте, що немає ніяких вимог по розташуванню колонок міток, директив, коментарів і команд.

Асемблер AVRASM транслює вихідний код в об'єктний код.

Згенерований об'єктний код використовується для симулятора AVR Studio, атакож для будь-якого іншого симулятора (емулятора). Асемблер також генерує код, який може бути безпосередньо завантажений в мікроконтролер.

Директиви асемблера

Директиви використовуються для керування розподілом пам'яті, визначенням макросів, ініціалізації пам'яті, змінних та ін. Директиви не транслюються в машинний код.

Перелік деяких директив асемблера AVR представлений в таблиці.

Директива	Опис
BYTE	Присвоїти байт змінній
CSEG	Сегмент коду
DB	Визначити байтові константи
DEF	Присвоїти символічне ім'я регістру

DSEG	Сегмент даних
DW	Визначити 2-ох байтову константу
EQU	Присвоїти символічному імені вираз
ESEG	EEPROM сегмент
INCLUDE	Ввімкнути файл
ORG	Встановити адресу програми

Приклад використання директив:

```
.include "m8515def.inc"          ; підключити файл опису
                                 ; для мікроконтролера ATmega8515
.DEF temp=r16                    ; присвоїтисимволічнеім'я temp
                                 ; регістру r16
.DEF counter=r17                 ; присвоїти символічне ім'я counter
                                 ; регістру r17
.ORG 0                           ; встановитидану адресу в $0
rjmp RESET                       ; перейти на мітку RESET
RESET:                           ; мітка RESET
;=================================================================
; ІНІЦІАЛІЗАЦІЯ
;=================================================================
ldi temp, LOW(RAMEND)            ; завантажити в регістр temp молодший байт
                                 ; слова RAMEND
out SPL, temp                    ; ініціалізація молодшого байта вказівника
                                 ; стека
ldi temp, HIGH(RAMEND)           ; завантажити в регістр temp старший байт
                                 ; слова RAMEND
out SPH, temp                    ; ініціалізація старшого байта вказівника
                                 ; стека
clr counter                      ; очистити регістр counter
ldi r30,LOW(var1)                ; завантажити в Z (r30) молодший байт var1
ldi r31,HIGH(var1)               ; завантажити в Z (r31) старший байт var1
sei                              ; дозволити переривання
;=================================================================
=
; ОСНОВНОЙ ЦИКЛ
;=================================================================
=
MainLoop:                        ; мітка основного циклу
inc counter                      ; інкримінуативміст регістру
                                 ; counter
rjmp MainLoop                    ; перейти на мітку MainLoop
;=================================================================
=
.DSEG                            ; початок сегмента даних
var1: .BYTE 1                    ; присвоїти байт 1 змінної var1
```

Інструкції (команди) мікроконтролера

Мікроконтролери серії AVR мають набір з більш ніж 130 інструкцій. Всі інструкції можна розділити на п'ять типів:

- арифметичні і логічні;

- інструкції розгалуження;
- інструкції пересилання даних;
- бітові інструкції та інструкції тестування бітів;
- інструкції керування мікроконтролером.

Основні команди (інструкції) мікроконтролерів серії AVRпредставлені в лабораторній роботі №4 (докладний опис всіх інструкцій можна знайти в керівництві користувача фірми ATMEL[InstructionSet.pdf]).

Асемблер не чутливий до регістру, тобто рядки

```
DEF temp=r16
DEF TEMP=R16
```

ідентичні.

Вирази, операнди і операції

Асемблер AVRASM включає в себе математичні вирази, що визначають константи. Вирази можуть містити операнди, оператори і функції.

Операнди

В асемблері AVRASM можуть бути використані наступні операнди:

- визначені користувачем мітки, які повертають значення адреси пам'яті того місця, де вони поставлені;
- визначені користувачем за допомогою директиви SET змінні;
- визначені користувачем за допомогою директиви EQU константи;
- цілі константи, які можуть бути представлені в одному з наступних форматів:
- десятковий: 10, 255;
- шістнадцятковий: 0x0a, $ 0A, 0xff, $ ff;
- двійковий: 0b00000000, 0b01010111;
- восьмеричний (упереджені нулем): 010, 077;
- PC - поточне значення програмного лічильника;

Оператори

Асемблер AVRASM підтримує 20 операторів, перелік яких представлений в таблиці в порядку пріоритетності:

Символ	**Опис**
!	Логічне «Ні»
~	Побітове «Ні»
-	Протилежний знак
*	Множення
/	Розподіл
+	Додавання
-	Віднімання

<<	Зрушення вліво
>>	Зсув вправо
<	Менше ніж
<=	Менше або дорівнює
>	Більше ніж
> =	Більше або дорівнює
==	Так само
! =	Не дорівнює
&	Побітове «І»
^	Побітова операція «Що виключає АБО»
|	Побітове «АБО»
&&	Логічне «І»
||	Логічне «АБО»

Функції

Асемблер AVRASM підтримує такі функції:

LOW (вираз) - повертає молодший байт виразу;

HIGH (вираз) - повертає старший байт виразу;

BYTE2 (вираз) - аналогічна функції HIGH;

BYTE3 (вираз) - повертає третій байт виразу;

BYTE4 (вираз) - повертає четвертий байт виразу;

LWRD (вираз) - повертає 0-15 біти виразу;

HWRD (вираз) - повертає 16-31 біти виразу;

PAGE (вираз) - повертає 16-21 біти виразу;

EXP2 (вираз) - повертає 2 в ступені виразу;

LOG2 (вираз) - повертає цілу частину двійкового логарифму виразу.

Порядок виконання роботи

1. Запустіть програмний симулятор AVR Studio, створіть новий проект з назвою "Assembler" на одному з доступних локальних дисків. В якості платформи для компілювання виберіть "AVR Simulator", а в якості пристрою - мікроконтролер "AT90S8535".

2. Скопіюйте в редактор коду наступний текст програми:

```
; =====================================================================
.include "8535def.inc"                    ; Підключити файл опису для AT90S8535
; =====================================================================
.def temp = r16                           ; Визначити змінну temp
; =====================================================================
; =====================================================================
;                          Опис векторів переривань
; =====================================================================
```

```
; ---------------------------------------------- -------------------------
; Вектор скидання
; ---------------------------------------------- -------------------------
.org 0                                          : Розмістити код за адресою $ 0
rjmp RESET                                      ; Перейти на мітку RESET
; ---------------------------------------------- -------------------------
; Вектор переривання по переповнення таймера / лічильника 1
; ---------------------------------------------- -------------------------
.org $ 008                                      ; Розмістити код за адресою $ 008
rjmp TIMER1_Overflow                            ; Перейти на мітку TIMER1_Overflow
; ---------------------------------------------- -------------------------
; Вектор переривання по завершенню АЦ-перетворення
; ---------------------------------------------- -------------------------
.org $ 00E                                      ;Розмістити код за адресою $ 00E
rjmp Conversion_Complete                        ; Перейти на мітку Conversion_Complete
; ============================================================================
; ============================================================================
; Початок програми
; ============================================================================
.org $ 20                                       ;Розмістити код за адресою $ 20
RESET:                                          ; Мітка RESET
; ---------------------------------------------- -------------------------
; Ініціалізація показника стека
; ---------------------------------------------- -------------------------
ldi temp, HIGH (RAMEND)
out sph, temp
ldi temp, LOW (RAMEND)
out spl, temp
; ---------------------------------------------- -------------------------
; Ініціалізація АЦП
; ---------------------------------------------- -------------------------
ldi temp, 0b00000111
out admux, temp                                 ; Вибрати канал АЦП №7
ldi temp, 0b10001000
out adcsr, temp                                 ; Дозволити роботу АЦП і переривання по
                                                ; завершенню перетворення
; ---------------------------------------------- -------------------------
; Ініціалізація універсального асинхронного приймально-передавача
; ---------------------------------------------- -------------------------
ldi temp, (1 << TXEN)
out ucr, temp                                   ; Дозволити роботу передавача
ldi temp, $ 19
out ubrr, temp                                  ; Вибрати швидкість передачі 19200 бод / с
; ---------------------------------------------- -------------------------
; Ініціалізація таймера / лічильника 1
; ---------------------------------------------- -------------------------
ldi temp, 0b00000010
out tccr1b, temp                                ; Запустити TC1 від основного генератора тактових
                                                ; імпульсів з розподілом частоти на 8
ldi temp, (1 << TOIE1)
out timsk, temp                                 ; Дозволити переривання по переповненню TC1
; ---------------------------------------------- -------------------------
sei                                             ; Дозволити глобальні переривання
; ============================================================================
; ============================================================================
```

```
; ОСНОВНИЙ ЦИКЛ
; =====================================================================
LOOP:                                   ; Основний цикл порожній, так як робота програми
                                        ;заснована на
rjmp LOOP                               ; використанні переривань
; =====================================================================
; =====================================================================
;                   Процедура обробки переривання таймера / лічильника 1
; =====================================================================
.org $ 50                               ;Розмістити код за адресою $ 50
TIMER1_Overflow:                        ; Мітка TIMER1_Overflow
sbi adcsr, $ 6                          ; Запустити аналого-цифрове перетворення
reti                                    ; Повернутися з підпрограми і дозволити
                                        ; переривання
; =====================================================================
; =====================================================================
;                   Процедура обробки переривання АЦП
; =====================================================================
.org $ 80                               ;Розмістити код за адресою $ 80
Conversion_Complete:                    ; Метка Conversion_Complete
in temp, adcl                           ; Зчитати в temp молодший байт результату
                                        ; АЦ-перетворення
out udr, temp                           ; Відправити по UART молодший байт результату
                                        ; АЦ-перетворення
Wait_For_Transmitt:                     ; Метка Wait_For_Transmitt
sbis usr, UDRE                          ; Регістр даних передавача порожній?
rjmp Wait_For_Transmitt                 ; Якщо не порожній перейти на мітку
                                        ; Wait_For_Transmitt
                                        ; Якщо порожній продовжити виконання програми
in temp, adch                           ; Зчитати в temp старший байт результату
                                        ; АЦ-перетворення
out udr, temp                           ; Відправити по UART старший байт результату
                                        ; АЦ-перетворення
reti                                    ; Повернутися з підпрограми і дозволити
                                        ; переривання
; =====================================================================
```

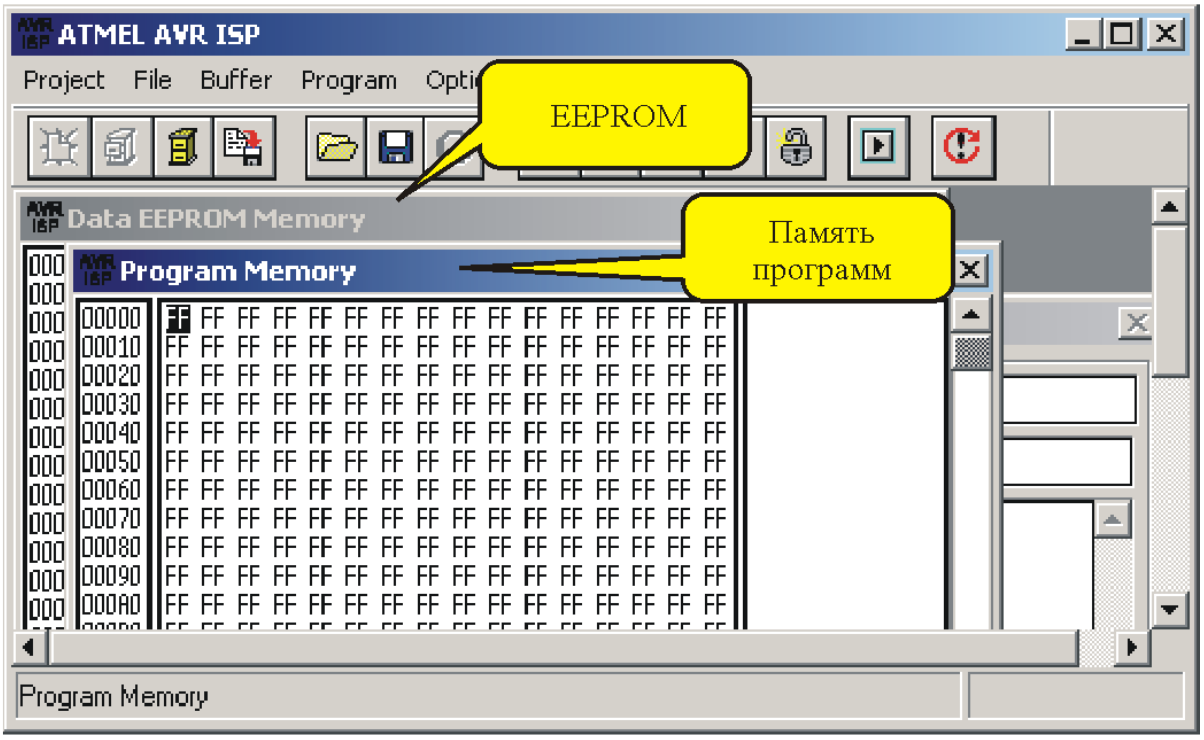
3. Скомпілюйте даний код, натиснувши клавішу F7. В області інформації з'являться повідомлення про хід компіляції.

Простежте за тим, щоб компіляція була завершена без помилок, в іншому випадку виправте помилки і скомпілюйте проект заново. Занесіть в таблицю результати про розподіл пам'яті мікроконтролера:

Область	Код, байт	Дані, байт	Використано, байт	Розмір, байт	Використано, %
Пам'ять програм (.cseg) ОЗУ (.dseg) EEPROM (.eseg)					

4. Запустіть внутрішньо-системний програматор Atmel AVR ISP.

5. Створіть новий проект, вибравши мікросхему AT90S8535.



6. Виберіть область пам'яті програм і завантажте в нього двійковий файл "Assembler.hex", отриманий в результаті компіляції проекту (File -> Load). В область пам'яті EEPROM завантажте двійковий файл "Assembler.eep", розміщений в тій же папці.

7. Розшукайте в пам'яті програм коди команди **rjmp** ($ Cx xx, x - число від $ 0 до $ F) і заповніть таблицю адрес команд **rjmp**.

№	Рядок коду	Адреса в AVR Studio	Адреса в AVR ISP
1	rjmp RESET	$ 00	$ 00
2	rjmp TIMER1_Overflow	…	…
3	rjmp Conversion_Complete	...	...
4	rjmp Wait_For_Transmitt	...	...

8. Проведіть порівняння місць розташування команд rjmp в AVR Studio і AVR ISP. Зробіть висновки, що пояснюють відмінність адрес.

9. Визначте адреси таблиць даних DATA_TABLE_1, DATA_TABLE_2 в пам'яті програм, DATA_TABLE_3, DATA_TABLE_4 вEEPROM і зробіть висновки про вплив директиви .org на розташування даних в різних видах пам'яті.

Контрольні питання

1. З чого складається код асемблера AVRASM? Формат ліній коду. Формат коментарів.
2. Директиви асемблера.
3. Інструкції мікроконтролера. Типи інструкцій.
4. Операнди, формати даних.
5. Оператори і функції.

Лабораторна робота №4. Набори команд мікроконтролерів серії AVR

Мета роботи: вивчити набір команд (інструкцій) мікроконтролерів серії AVR

Теоретичні відомості

Основні команди (інструкції) мікроконтролерів серії AVR представлені в таблицях (докладний опис всіх інструкцій можна знайти в керівництві користувача фірми ATMEL [InstructionSet.pdf]).

Команди представлені у вигляді таблиць, що мають такі стовпці:

- **мнемоніка** – символічна назва команди, що спрощує її запам’ятовування. При компіляції на асемблері мнемоніка замінюється на відповідним їй код;
- **операнди** - операнди, задіяні при виконанні команди;
- **опис** - опис команди;
- **операція** - арифметичне (логічне) представлення команди;
- **прапори** - біти регістра статусу, на які впливає дана команда при виконанні;
- **цикли** - кількість циклів, необхідне для виконання команди.

Формат операндів, використаних в переліку команд, наведено нижче:

Rd - регістр-одержувач в регістровому файлі;

Rr - регістр-джерело;

b - константа (0-7), може бути математичний вираз;

s - константа (0-7), може бути математичний вираз;

P - константа (0-31 / 63), може бути математичний вираз;

K6 - константа (0-63), може бути математичний вираз;

K8 - константа (0-255), може бути математичний вираз;

K - константа (діапазон значень залежить від інструкції), можебути математичний вираз;

q - константа (0-63), може бути математичний вираз;

X, Y, Z - непрямі адресні регістри (X = R27: R26, Y = R29: R28,Z = R31: R30).

Всі інструкції мікроконтролерів серії AVR можна розділити нап'ять типів:

- арифметичні і логічні;
- інструкції розгалуження;
- інструкції пересилання даних;
- бітові інструкції та інструкції тестування бітів;
- інструкції управління мікро контролером.

Розглянемо докладніше кожен з п'яти типів інструкцій.

Арифметичні і логічні

До даного типу інструкцій відносяться інструкції множення, додавання, віднімання, логічних операцій "І", "АБО", "Що виключає АБО",інструкції

установки і очищення регістрів і бітів в регістрах, інструкції інкрементування, декрементування тощо.

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
арифметичні та логічні					
ADD	Rd,Rr	Додати без переносу	Rd=Rd+Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Додати з переносом	Rd=Rd+Rr+C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логічне «І»	Rd=Rd·Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне «І» з безпосереднім значенням	Rd=Rd·K8	Z,N,V,S	1
OR	Rd,Rr	Логічне «АБО»	Rd=Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логічне «АБО» з безпосереднім значенням	Rd=Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Виняткове АБО	Rd=Rd EOR Rr	Z,N,V,S	1
SBR	Rd,K8	Встановити біт в регістрі	Rd=RdVK8	Z,C,N,V,S	1
CBR	Rd,K8	Очистити біт в регістрі	Rd=Rd·($FFK8)Z ,C,N,V,S	1	1
INC	Rd	Інкрементувати регістр	Rd=Rd+1	Z,N,V,S	1
DEC	Rd	Декрементувати регістр	Rd=Rd-1	Z,N,V,S	1
CLR	Rd	Очистити ругістр	Rd=0	Z,C,N,V,S	1
SER	Rd	Встановити регістр	Rd=$FF	None	1
MUL	Rd,Rr	Помножити беззнакові	R1:R0=Rd*Rr	Z,C	2
MULS	Rd,Rr	Помножити зі знаком	R1:R0=Rd*Rr	Z,C	2
MULSU	Rd,Rr	Помножити знакові з без знаковими	R1:R0=Rd*Rr	Z,C	2

ADD - Додати без переносу

Опис:

Додає вміст двох регістрів без урахування прапора перенесення C і поміщає результат в регістр Rd.

Синтаксис:

ADD Rd, Rr

Приклад:

```
add r1,r2                ;додати r2 к r1 (r1=r1+r2)
add r28,r28              ;додати r28 до самого себе(r28=r28+r28)
```

ADC - додавання з перенесенням

Опис:

Складає вміст двох регістрів, а також вміст прапора перенесення C і поміщає результат в регістр Rd.

Синтаксис:

ADC Rd, Rr

Приклад:

```
            ; Складання слова
add r2, r0     ; Скласти молодші байти
adc r3, r1     ; Скласти старші байти з перенесенням
```

AND - логічне «І»

Опис:

Виконує логічне «І» між вмістом двох регістрів і поміщає результат в регістр Rd.

Синтаксис:

AND Rd, Rr

Приклад:

```
and r2, r3          ; Побітове «І» між r2 і r3, результат в r2
ldi r16,1           ; Встановити бітову маску 0000 0001 в r16
and r2, r16                 ; Маскувати нульовий біт в r2
```

ANDI - Логічне «І» з безпосереднім значенням

Опис:

Виконує логічне «І» між вмістом регістра Rd і констант і поміщає результат в регістр Rd.

Синтаксис:

ANDI Rd, K

Приклад:

```
andi r17, $ 0F          ; Очистити старший напівбайт в r17
andi r18, $ 10          ; Очистити всі біти крім 4-го в r18
andi r19, $ AA          ; Очистити непарні біти в r19
```

SBR - Встановити біт (и) в регістрі

Опис:

Встановити зазначені біти в регістрі Rd. Виконує операцію логічного додавання між вмістом регістра Rd і константою K, поміщуючи результат в Rd.

Синтаксис:

SBR Rd, K

Приклад:

```
sbr r16,3           ; Встановити біти 0 і 1 в r16
sbr r17, $ F0           ; Встановити старший напівбайт в r17
```

INC - інкрементувати регістр

Опис:

Додає до вмісту регістра Rd одиницю і поміщає результатв Rd.

Синтаксис:

INC Rd

Приклад:

```
clr r22                 ; Очистити r22
loop:                   ; Мітка
inc r22                 ; інкрементувати r22
cpi r22, $ 4F           ; Порівняти r22 з константою $ 4F
```

```
brne loop          ; Перейти на мітку loop якщо не дорівнює
                   ; Якщо r22 дорівнює $ 4f виконуються операції йдуть
                   ; далі
```

SER - Встановити регістр

Опис:

Завантажує в регістр Rd константу $ FF.

Синтаксис:

SER Rd

Приклад:

```
clr r16                ; Очистити r16
ser r17                ; Встановити r17
out $ 18, r16          ; Записати нулі за адресою $ 18 (порт B)
nop                    ; Затримка (немає операції)
out $ 18, r17          ; Записати одиниці за адресою $ 18 (порт B)
```

MUL - Помножити беззнакові

Опис:

Примножує два беззнакових 8-розрядних числа, 16-розрядний результат поміщається в пару регістрів R1: R0.

Синтаксис:

MUL Rd, Rr

Приклад:

```
mul r5, r4         ; Перемножити беззнакову r5 і r4
movw r4, r0              ; Скопіювати результат з r1: r0 в r5: r4
```

Інструкції розгалуження

До інструкцій розгалуження відносяться команди умовних і безумовних переходів, команди виклику і повернення з підпрограм і команди порівняння.

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
інструкції розгалуження					
RJMP	k	Перейти відносно	PC=PC+k+1	None	2
RCALL	k	Викликати підпрограму відносно	STACK=PC+1,PC=PC+k+1	None	3/4*
RET	None	Повернутися з підпрограми	PC=STACK	None	4/5*
RETI	None	Повернутись з переривання	PC=STACK	I	4/5*
CPI	Rd,K8	Порівняти з константою	Rd-K	Z,C,N,V,H,S	1
SBRC	Rr,b	Пропустити, якщо біт в реєстрі очищений	if(Rr(b)==0) PC=PC+2or3	None	1/2/3
SBRS	Rr,b	Пропустити, якщо біт в реєстрі	if(Rr(b)==1) PC=PC+2or3	None	1/2/3

		встановлений			
SBIC	P,b	Пропустити, якщо біт в реєстрі I/O очищений	if(I/O(P,b)==0) PC=PC+2or3	None	1/2/3
SBIS	P,b	Пропустити, якщо біт в реєстрі I/O встановлений	if(I/O(P,b)==1) PC=PC+2or3	None	1/2/3
BREQ	k	Перейти, якщо рівно	if(Z==1) PC=PC+k+1	None	1/2
BRNE	k	Перейти, якщо не рівно	if(Z==0) PC=PC+k+1	None	1/2

RJMP - Перейти відносно

Опис:

Виконує відносний перехід за адресою PC-2K + 1 і PC + 2K (слів). В асемблері замість зсуву адреси переходу використовуються мітки.

Для AVR мікроконтролера, обсяг пам'яті якого не перевищує 4K слів (8Кб), дана команда може здійснювати переходи по всьому простору програмної пам'яті з будь-якої адреси.

Синтаксис:

RJMP k

Приклад:

```
cpi r16, $ 42          ; Порівняти r16 с $ 42
brne error             ; Перейти якщо r16 не дорівнює $ 42
rjmp ok          ; Перейти якщо r16 дорівнює $ 42
error: add r16, r17    ; Скласти r16 і r17
inc r16                ; інкрементувати r16
ok: nop          ; Метка для rjmp (немає операції)
```

RCALL - Відносний виклик підпрограми

Опис:

Виконує відносний виклик підпрограми за адресою PC-2K + 1 іPC + 2K (слів). Вихідна адреса PC + 2 зберігається в стеку. В асемблері замість зміщення адреси переходу використовуються мітки. Для AVR мікроконтролера, обсяг пам'яті якого не перевищує 4K слів (8Кб), дана команда може здійснювати виклик по всьому простору програмної пам'яті збудь-якої адреси.

Синтаксис:

RCALL k

Приклад:

```
rcall routine          ; Виклик підпрограми
...
routine:          ; Початок підпрограми
push r14          ; Зберегти r14 в стеці
```

```
...                 ; Команди в підпрограмі
pop r14             ; Відновити r14
ret                 ; Повернення з підпрограми
```

RET - Повернутися з підпрограми

Опис:

Здійснює повернення з підпрограми за адресою, який буде збережений при виконанні команди RCALL.

Синтаксис:

RET

Приклад:

```
rcall routine           ; Виклик підпрограми
...
routine:            ; Початок підпрограми
push r14            ; Зберегти r14 в стеці
...                 ; Команди в підпрограмі
pop r14             ; Відновити r14
ret                 ; Повернення з підпрограми
```

RETI - Повернутися з переривання

Опис:

Здійснює повернення з переривання за адресою, який буде збережений при вході в переривання. Також встановлюється прапор глобального дозволу переривання. Регістр статусу не зберігається автоматично при вході в переривання і не відновлюється при поверненні з переривання.

Збереження регістра статусу має, при необхідності, здійснюватиметься програмою.

Синтаксис:

RETI

Приклад:

```
...
extint: push r0         ; Зберегти r0 в стеці
...
pop r0              ; Відновити r0 з стека
reti                ; Повернуться з переривання
```

CPI - Порівняти з константою

Опис:

Виконує порівняння Rd і константи. Вміст регістра Rd залишається без змін. Після даної команди використовуються команди розгалуження.

Синтаксис:

CPI Rd, k

Приклад:

```
cpi r19,3           ; Порівняти r19 з трійкою
brne error              ; Перейти якщо r19 не дорівнює 3
...
```

```
error: nop          ; Метка для переходу
```

SBRC - Пропустити якщо біт в регістрі очищений

Опис:

Команда тестує біт в регістрі і пропускає наступну команду,якщо біт очищений.

Синтаксис:

SBRC Rr, b

Приклад:

```
sub r0, r1              ; Відняти r1 з r0
sbrc r0,7               ; Пропустити наступну команду, якщо біт 7 в r0 очищений
sub r0, r1              ; Виконується тільки якщо біт 7 в r0 встановлений
...                     ; Продовження
```

BREQ - Перейти якщо одно

Опис:

Умовний відносний перехід. Тестує біт Z в регістрі статусу і виконує відносний перехід, якщо Z встановлений. Дана команда використовується безпосередньо після виконання команд порівняння, арифметичних і логічних команд, а також після інших команд, які надають вплив на прапор Z.

Синтаксис:

BREQ k

Приклад:

```
cp r1, r0           ; Порівняти регістри r1 і r0
breq equal              ; Перейти на мітку equal якщо r0 = r1
...
equal: nop              ; Мітка для переходу
```

Інструкції пересилання даних

До інструкцій пересилання даних відносяться команди копіювання даних, безпосереднього і непрямого завантаження даних, команди роботи зі стеком.

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
інструкції пересилання даних					
MOV	Rd,Rr	Копіювання регістру	Rd=Rr	None	1
LDI	Rd,K8	Завантажити безпосередньо	Rd=K	None	1
LDS	Rd,k	Завантажити безпосередньо з ОЗУ	Rd=(k)	None	2*
LD	Rd,X	Завантажити опосередковано	Rd=(X)	None	2*
LD	Rd,Y	Завантажити опосередковано	Rd=(Y)	None	2*
LD	Rd,Z	Завантажити опосередковано	Rd=(Z)	None	2*

STS	k,Rr	Записати безпосередньо в ОЗУ	(k)=Rr	None	2*
ST	X,Rr	Записати опосередковано	(X)=Rr	None	2*
ST	Y,Rr	Записати опосередковано	(Y)=Rr	None	2*
ST	Z,Rr	Записати опосередковано	(Z)=Rr	None	2
LPM	None	Завантажити байт з пам'яті програм	R0=(Z)	None	3
IN	Rd,P	Завантажити з регістру I/O	Rd=P	None	1
OUT	P,Rr	Записати в регістр I/O	P=Rr	None	1
PUSH	Rr	Помістити регістр в стек	STACK=Rr	None	2
POP	Rd	Завантажити регістр із стеку	Rd=STACK	None	2

MOV - Копіювати регістр

Опис:

Команда копіює вміст регістра Rr в регістр Rd. Вміст регістра Rr не змінюється.

Синтаксис:

MOV Rd, Rr

Приклад:

```
mov r16, r0                  ; Копіювати r0 в r16
call check                   ; Виклик підпрограми
...
check: cpi r16, $ 11   ; Порівняти r16 з числом $ 11
...
Ret                          ; Повернення з підпрограми
```

LDI - Завантажити безпосередньо

Опис:

Команда завантажує 8-бітну константу в один з регістрів R16: R31.

Синтаксис:

LDI Rd, K

Приклад:

```
clr r31               ; Очистити старший байт покажчика Z (реєстр r31)
ldi r30, $ F0                ; Завантажити в молодший байт покажчика Z
                    ; (Реєстр r30) константу $ F0
Lpm                          ; Завантажити байт з програмної пам'яті на який
                    ; Вказує покажчик Z
```

LDS - Завантажити безпосередньо з ОЗУ

Опис:

Команда завантажує в регістр Rd байт з простору пам'яті. Для мікроконтролерів, що мають ОЗУ простір пам'яті включає в себе регістри загального призначення (регістровий файл), простір введення-виведення,

і внутрішнє ОЗУ (а також зовнішнє ОЗУ, при його наявності). Дана команда доступна не у всіх пристроях.

Синтаксис:

LDS Rd, k

Приклад:

```
lds r2, $ FF00                ; Завантажити в r2 вміст комірки пам'яті з
                    ; адресою $ FF00
add r2, r1                    ; Додати r1 до r2
sts $ FF00, r2                ; Записати r2 в комірку пам'яті з адресою $ FF00
```

LD - Завантажити побічно з ОЗУ

Опис:

Команда побічно завантажує в регістр Rd байт з простору пам'яті на який вказує покажчик. Для мікроконтролерів, що мають ОЗУ простір пам'яті включає в себе регістри загального призначення (регістровий файл), простір введення-виведення, і внутрішнє ОЗУ (а також зовнішнє ОЗУ, при його наявності). Команда може використовувати 3 16-розряднихпокажчика X, Y, Z. Покажчик може бути залишений без змін після виконання команди, може бути постінкрементований або предекрементований.

Не всі різновиди даної команди доступні у всіх пристроях.

Синтаксис:

LD Rd, X

LD Rd, X +

LD Rd, -X

Приклад:

```
clr r27                       ; Очистити старший байт покажчика X
ldi r26, $ 60                 ; Завантажити в молодший байт покажчика X число $ 60
ld r0, X +                  ; Завантажити в r0 вміст комірки пам'яті за адресою $ 60,
                    ; Після виконання команди вміст покажчика X
                    ; інкрементується
ld r1, X                      ; Завантажити в r0 вміст комірки пам'яті за адресою $ 61
ldi r26, $ 63                 ; Завантажити в молодший байт покажчика X число $ 63
ld r2, X                      ; Завантажити в r0 вміст комірки пам'яті за адресою $ 63
ld r3, -X                   ; Завантажити в r0 вміст комірки пам'яті за адресою $ 62
                    ; (Покажчик X декрементований до виконання команди)
```

IN - Завантажити в регістр байт з простору введення-виведення

Опис:

Завантажує байт з простору вводу-виводу (порти, таймери, конфігураційні регістри і ін.) в регістр загального призначення r0: r31.

Синтаксис:

IN Rd, A

Приклад:

```
in r25, $ 16            ; Вважати в r25 дані на порту B (реєстр $ 16)
cpi r25,4           ; Порівняти зчитане з 4
breq exit           ; Перейти якщо r25 = 4
...
exit: nop           ; Мітка для переходу
```

PUSH - Помістити регістр в стек

Опис:

Інструкція зберігає вміст регістру Rr в стеці. Покажчик стека постдекрементується.

Дана інструкція доступна не у всіх пристроях.

Синтаксис:

PUSH Rr

Приклад:

```
call routine        ; Виклик підпрограми
...
routine: push r14   ; Помістити r14 в стек
push r13            ; Помістити r13 в стек
...                 ; Код підпрограми
pop r13                 ; Відновити r13
pop r14                 ; Відновити r14
ret                 ; Повернення з підпрограми
```

Бітові інструкції та інструкції тестування бітів

До даного типу інструкцій відносяться інструкції логічного і арифметичного зсувів, інструкції установки бітів в регістрах введення-виведення та ін.

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
Бітові інструкції та інструкції тестування бітів					
LSL	Rd	Логічний зсув вліво	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Логічний зсув вправо	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Зсув вліво через перенесення	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Зсув вправо через перенесення	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
SWAP	Rd	Змінити нібли місцями	Rd(3..0)=Rd(7..4), Rd(7..4)=Rd(3..0)	None	1
SBI	P,b	Встановити біт в регістрі I/O	I/O(P,b)=1	None	2
CBI	P,b	Очистити біт в регістрі I/O	I/O(P,b)=0	None	2

SEI	None	Дозволити переривання	I=1	I	1
CLI	None	Заборонити переривання	I=0	I	1

LSL - Логічний зсув вліво

Опис:

Зрушує всі біти в Rd на одну позицію вліво. Біт 0 очищається. Біт 7 завантажується у прапор C регістра статусу. Операція LSL здійснює без знакове множення на 2.

Синтаксис:

LSL Rd

Приклад:

```
add r0, r4          ; Додати до r0 r4
lsl r0              ; Помножити r0 на 2
```

ROL - Зрушення вліво через перенос

Опис:

Зрушує все біти в Rd на одну позицію вліво. Біт 0 завантажується з прапора перенесення C регістра статусу. Біт 7 завантажується у прапор C регістра статусу. Операція ROL використовується спільно з LSL для множення на 2 багато-байтних чисел.

Синтаксис:

ROL Rd

Приклад:

```
lsl r18             ; Помножити r19: r18 на 2
rol r19             ; r19: r18 знакова або без знакове ціле число
```

SWAP - Поміняти напівбайтів місцями

Опис:

Міняє місцями старший і молодший півбайт регістра Rd.

Синтаксис:

SWAP Rd

Приклад:

```
inc r1              ; інкрементувати r1
swap r1         ; Поміняти місцями старший і молодший півбайт регістра r1
inc r1              ; інкрементувати старший напівбайт r1
swap r1         ; Поміняти назад місцями старший і молодший півбайт регістра r1
```

SBI - Встановити біт в регістрі введення-виведення

Опис:

Встановлює зазначений біт в регістрі введення-виведення. Дана інструкція працює тільки з регістрами, що мають адреси від 0 до 31.

Синтаксис:

SBI A, b

Приклад:

```
out $ 1E, r0          ; Записати адресу EEPROM
sbi $ 1C, 0       ; Встановити біт читання в EECR
in r1, $ 1D       ; Прочитати дані з EEPROM
```

SEI - Дозволити переривання

Опис:

Встановлює прапор глобального дозволу переривання I в регістрістатусу.

Синтаксис:

SEI

Приклад:

```
sei               ; Дозволити переривання
sleep                 ; Перейти в режим SLEEP і чекати переривання
```

Інструкції керування мікроконтролером

Дані інструкції призначені для керування мікроконтролером.

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
Інструкції керування мікроконтролером					
NOP	None	Нема операції	None	None	1
SLEEP	None	Встановити режим SLEEP	See instruction manual	None	1
WDR	None	Скинути сторожовий таймер	See instruction manual	None	1

NOP - Немає операції

Опис:

Не виконує ніяких операцій, займає один цикл.

Синтаксис:

NOP

приклад:

```
clr r16               ; Очистити r16
ser r17               ; Встановити r17
out $ 18, r16         ; Записати нулі в регістр введення-виведення $ 18 (порт B)
nop                   ; Затримка в один цикл для встановлення порту
out $ 18, r17         ; Записати одиниці в регістр введення-виведення $ 18 (порт B)
```

SLEEP - Встановити режим SLEEP

Опис:

Переводить мікроконтролер в визначений користувачем режимзниженого енергоспоживання.

Синтаксис:

SLEEP

Приклад:

```
mov r0, r11           ; Копіювати r11 в r0
ldi r16, $ 20         ; Дозволити SLEEP режим
```

```
out MCUCR, r16
sleep                    ; Перевести мікроконтролер в режим SLEEP
```

WDR - Скинути сторожовий таймер

Опис:

Дана інструкція скидає сторожовий таймер. При запущеному сторожовому таймері дана інструкція повинна виконуватися через обмежений проміжок часу, встановлений дільником частоти таймера.

Синтаксис:

WDR

Приклад:

```
wdr              ; Скинути сторожовий таймер
```

Порядок виконання роботи

1. Ознайомитися з переліком інструкцій мікроконтролерів сеpіїAVR.

2. Розрахувати вихідні дані для виконання завдання по формулам:

$$a = 9*N + I,\ b = 8*N + I,$$

де N - порядковий номер за журналом;

I - індекс групи, (видає викладач).

3. Покроково виконати представлену програму, заповнюючи таблицю наведену нижче.

```
ldi r16, a
ldi r17, b
inc r16
lsl r16
swap r16
sbrc r16, $ 1
rjmp case_2
case_1:
add r16, r17
ori r16, $ aa
and r16, r17
sbr r16, $ 3
rjmp exit
case_2:
cbr r16, $ 3
ori r16, $ 55
and r16, r17
add r16, r17
exit:
```

I=…	N=…	a=…	b=…
№ кроку	Команда яка виконується	Значення в r16	Опис команди
1			
…			

Приклад заповнення таблиці:

I=1	N=1	a=9*1+1=10	b=8*1+1=9
№ кроку	Команда яка виконується	Значення в r16	Опис команди
1	`ldi r16, a`	10	Завантажити в r16 число 10
2	`ldi r17, b`	10	Завантажити в r17число 9
3	`inc r16`	11	Інкрементувати r16
4	`lsl r16`	22	Логічний зсув вліво r16
5	`swap r16`	22	Змінити місцями старший та молодший полубайт
…	…	…	…

Контрольні питання

1. Поняття мнемоніки, операнда, прапора.
2. Типи інструкцій мікроконтролерів AVR.
3. Арифметичні і логічні інструкції.
4. Інструкції розгалуження.
5. Інструкції пересилки даних.
6. Бітові інструкції та інструкції тестування бітів.
7. Інструкції керування мікроконтролером.

Лабораторна робота №5. Дослідження режимів роботи портів введення-виведення для мікроконтролерів серії AVR

Мета роботи: ознайомитися з режимами роботи портів вводу-виводу (ПВВ) мікроконтролера ATmega8515, провести налаштування портів за вказаними даними.

Теоретичні відомості

Мікроконтролер ATmega8515 має чотири або п’ять ПВВ, в залежності від корпусу. Порти мають наступні назви: PORTA, PORTB, PORTC, PORTD, PORTE.

Здатність навантаження всіх ліній ПВВ достатня для безпосереднього керування світлодіодним індикатором. Всі лінії ПВВ володіютьможливістю програмного підключення вбудованого підтягуючого резистора. Також лінії ПВВ мають захисні діоди в напрямках як до напругиживлення (VCC) так і до загального проводу (GND). Еквівалентнаелектрична схема лінії ПВВ представлена на малюнку 6.1.

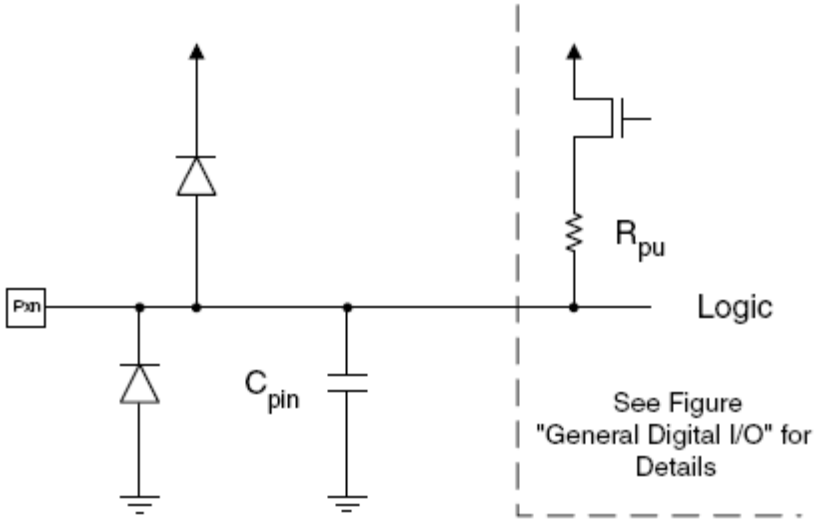


Рис 5.1 - Еквівалентна електрична схема лінії ПВВ

Для кожного з ПВВ зарезервовано три осередки пам'яті (x - ім'я порта):

- регістр даних - PORTx;
- регістр напрямку даних - DDRx;
- вивід порту - PINx.

Регістри даних і напрямку можуть читатися і записуватися, регістра PINx тільки для читання.

Установка біту PUD (Pull-up Disable) в регістрі SFIOR відключає підтягуючі резистори для всіх ліній всіх ПВВ.

Більшість ліній ПВВ мають альтернативні функції, пов'язаніз використанням таких пристроїв як універсальний синхронно-асинхронний приймач-передавач (USART), АЦП (ADC), таймери-лічильники(TC), послідовний периферійний інтерфейс (SPI), аналоговий компаратор (AC) і ін.

Конфігурація ліній портів вводу-виводу

Кожен біт DDxn регістра DDRx вибирає напрямок відповідної лінії порту x. Якщо біт DDxn встановлений, відповідна лініяпорту налаштована як вихід. Якщо біт DDxn очищений, відповідна лініяпорту налаштована як вхід.

Якщо біт регістра порту PORTxn встановлений і дана лінія портуналаштована як вхід, активізується відповідний підтягуючий резистор. Для виключення підтягуючого резистора повинен бути очищений бітPORTxn, або дана лінія порту повинна бути переналаштована як вихід.

Запис одиниці в біт PORTxn в стані, коли лінія порту налаштована як вихід, переводить лінію порту і відповідну ніжку мікросхеми в стан логічної одиниці. Запис нуля в біт PORTxn в стані, коли лінія порту налаштована як вихід, переводить лінію порту івідповідну ніжку мікросхеми в стан логічного нуля.

Всі можливі конфігурації ліній ПВВ представлені в таблиці5.1

Табл. 5.1 - Зміни ліній ПВВ

DDxn	PORTxn	PUD(SFIOR)	Вхід/вихід	Підтяжка	Коментар
0	0	x	Вхід	Ні	Третій стан
0	1	0	Вхід	Є	Лінія є джерелом струму
0	1	1	Вхід	Ні	Третій стан
1	0	x	Вихід	Ні	Логічний нуль
1	1	x	Вихід	Ні	Логічна одиниця

Незалежно від установок біта напрямки даних DDxn, логічний рівень може бути зчитаний безпосередньо з ніжки мікросхеми за допомогою бітів регістра PINx.

Керуючі регістри портів введення-виведення

Далі представлені структури керуючих регістрів порту A.Керуючі регістри всіх інших портів мають аналогічну структуру.Назви регістрів і бітів ПВВ мають відмінності в одній букві, яка відповідає назві порту: PORTA, PORTB, PORTC, DDRA, DDRB, DDRE,PINA, PINC, PIND і т.д.

Бит	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Чтение/Запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

Бит	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Чтение/Запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

Бит	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Чтение/Запись	R	R	R	R	R	R	R	R	
Начальное значение	неопределено			неопределено			неопределено		

Зовнішні переривання

Зовнішні переривання можуть бути згенеровані з використанням висновків мікросхеми INT0, INT1, INT2. Зауважте також те, що якщо переривання дозволено, воно може бути згенеровано навіть якщо висновки INT0, INT1, INT2 налаштовані як вихідні. Це дає можливість реалізації програмних генерацій переривань. Зовнішнє переривання може бути згенеровано спадаючим, зростаючим фронтами або низьким рівнем (на INT2 тільки фронтами). Тип переривання налаштовується в регістрі мікроконтролера (MCUCR - MCU Control Register) і розширеному керуючому регістрі мікроконтролера (EMCUCR - Extended MCU Control Register).

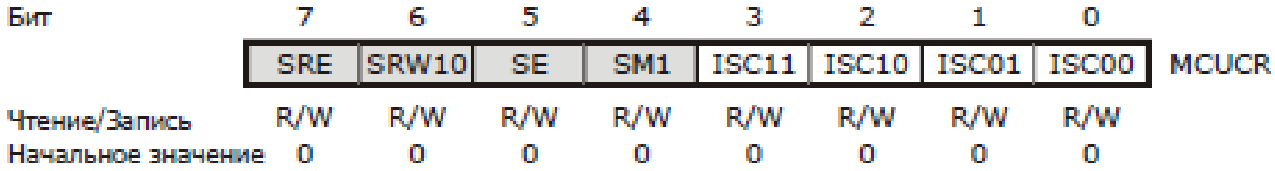
Якщо переривання допустиме і налаштоване на спрацьовування по рівню (можливо тільки для INT0, INT1), то переривання буде генеруватися до тих пір, поки на відповідному виводі мікросхеми буде присутній нульовий логічний рівень.

Спрацьовування переривання по фронтах на виводах INT0, INT1 вимагаєприсутності тактовихімпульсів, отже, воно не може бути згенеровано в сплячих режимах, в яких генератор тактових імпульсів зупинений. Переривання по рівню на висновках INT0, INT1, а також перериванняпо фронтах на виводах INT2 детектуются асинхронно, що дає можливість використання даних переривань для виведення мікроконтролера зсплячого режиму. Генератор тактових імпульсів зупинений у всіх сплячихрежимах за винятком режиму Idle.

Керуючий регістр мікроконтролера MCUCR

Керуючий регістр мікроконтролера MCUCR містить біти керування типом чутливості зовнішніх переривань INT0 і INT1, атакож загальними функціями

мікроконтролера. Розглянемо біти керування типом чутливості зовнішніх переривань.



• біти 3, 2 - ISC11, ISC10: Interrupt Sense Control 1 (керування чутливістю переривання INT1), біти 1 і 0

Зовнішнє переривання 1 активне, якщо встановлені біт глобальногопереривання в регістрі статусу (SREG) і відповідний маскуючий біт в регістрі GICR. Рівні та фронти, за якими генерується переривання INT1, представлені в таблиці 5.2.

Табл. 5.2 - Керування чутливістю INT1

ISC11	ISC10	Опис
0	0	Низький логічний рівень на INT1 генерує переривання
0	1	Будь-яка зміна логічного рівня на INT1 генерує переривання
1	0	Спадаючий фронт на INT1 генерує переривання
1	1	Зростаючий фронт на INT1 генерує переривання

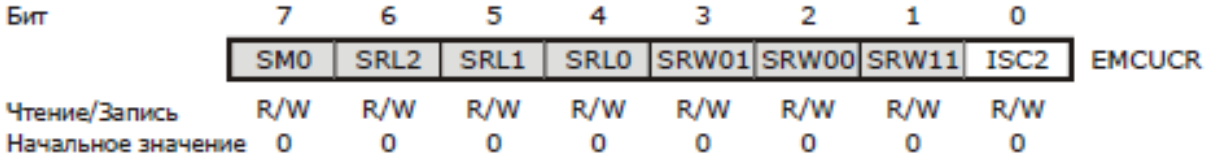
• біти 1, 0 - ISC01, ISC00: Interrupt Sense Control 0 (керування чутливістю переривання INT0), біти 1 і 0

Зовнішнє переривання 0 активне, якщо встановлені біт глобального переривання в регістрі статусу (SREG) і відповідний маскуючий біт в регістрі GICR. Рівні та фронти, за якими генерується переривання INT0, представлені в таблиці 5.3.

Табл. 5.3 - Керування чутливістю INT0

ISC01	ISC00	Опис
0	0	Низький логічний рівень на INT0 генерує переривання
0	1	Будь-яка зміна логічного рівня на INT0генерує переривання
1	0	Спадаючий фронт на INT0 генерує переривання
1	1	Зростаючий фронт на INT0 генерує переривання

Розширений керуючий регістр мікроконтролера EMCUCR



• біт 0 - ISC2: Interrupt Sense Control 2 (керування чутливістю переривання INT2)

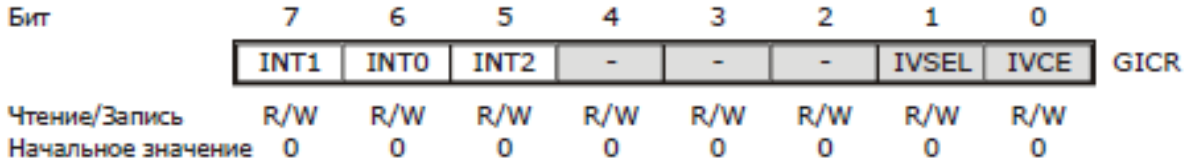
Асинхронне зовнішнє переривання 2 активне, якщо встановлені біт глобального переривання в регістрі статусу (SREG) і відповідний маскуючий біт в регістрі GICR. Фронти, за якими генерується переривання INT2, представлені в таблиці 5.4.

Табл. 5.4 - Управління чутливістю INT2

ISC2	Опис
0	Спадаючий фронт на INT2 генерує переривання
1	Зростаючий фронт на INT2 генерує переривання

Фронти на INT2 реєструються асинхронно.

Регістр керування перериваннями (GICR - General Interrupt Control Register)



• біт 7 - INT1: External Interrupt Request 1 Enable (зовнішнє переривання 1 активне)

Даний біт, разом з бітом глобального переривання I активізуєпереривання INT1.

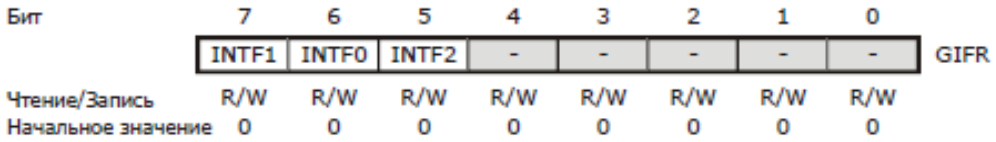
• біт 6 - INT0: External Interrupt Request 0 Enable (зовнішнє переривання 0 активне)

Даний біт, разом з бітом глобального переривання I активізуєпереривання INT0.

• біт 5 - INT2: External Interrupt Request 2 Enable (зовнішнє переривання 2 активне)

Даний біт, разом з бітом глобального переривання I активізуєпереривання INT2.

Регістр прапорів переривань (GIFR - General Interrupt Flag Register)



• біт 7 - INTF1: External Interrupt Flag 1 (прапор зовнішнього переривання 1)

Коли фронт або зміна логічного рівня на виводі INT1, призводить до генерації переривання, біт INTF1 встановлюється. Якщо біт глобального переривання I в регістрі статусу, а також біт дозволу переривання INT1 в регістрі GICR встановлені, мікроконтролер здійснює перехід на відповідний вектор переривання. Прапор зовнішнього переривання 1 INTF1 автоматично очищається після виконання підпрограми обробки переривання. Також прапор INTF1 може бути очищений записом в нього логічної одиниці. Даний прапор завжди очищений, якщо INT1 налаштований на спрацювання за рівнем.

• біт 6 - INTF0: External Interrupt Flag 0 (прапор зовнішнього переривання 0)

Коли фронт або зміна логічного рівня на виводі INT0, призводить до генерації переривання, біт INTF0 встановлюється. Якщо біт глобального переривання I в регістрі статусу, а також біт дозволу переривання INT0 в

регістрі GICR встановлені, мікроконтролер здійснює перехід на відповідний вектор переривання. Прапор зовнішнього переривання 0 INTF0 автоматично очищається після виконання підпрограми обробки переривання. Також прапор INTF0 може бути очищений записом в нього логічноїодиниці. Даний прапор завжди очищений, якщо INT0 налаштований на спрацювання за рівнем.

• біт 5 - INTF2: External Interrupt Flag 2 (прапор зовнішнього переривання 2)

Коли фронт або зміна логічного рівня на виводі INT2, призводитьдо генерації переривання, біт INTF2 встановлюється. Якщо біт глобальногопереривання I в регістрі статусу, а також біт дозволу переривання INT2 врегістрі GICR встановлені, мікроконтролер здійснює перехід на відповідний вектор переривання. Прапор зовнішнього переривання 2 INTF2 автоматично очищається після виконання підпрограми обробки переривання. Також прапор INTF2 може бути очищений записом в нього логічноїодиниці.

Робота з портами вводу-виводу

Всі лінії портів вводу-виводу можуть бути індивідуально налаштованіна введення і виведення. Налаштування роботи окремої лінії порту може бути змінено з використанням інструкцій SBI, CBI.

```
sbi DDRB, $ 2                ; встановити лінію 2 порту B на вивід
cbi PORTB, $ 2               ; встановити логічний нуль на лінію 2 порту B
nop                          , нема операції (затримка для встановлення порту)
nop                          ;
sbi PORTB, $ 2               ; встановити логічну одиницю на лінію 2 порту B
nop                          , нема операції (затримка для встановлення порту)
nop                          ;
cbi PORTB, $ 2               ; встановити логічний нуль на лінію 2 порту B
nop                          , нема операції (затримка для встановлення порту)
nop                          ;
```

В результаті виконання даного коду на виводі PB2 мікросхеми буде згенерований одиночний імпульс:

Робота з усіма лініями порту одночасно проводиться з використанням інструкцій IN, OUT.

```
ldi r16, $ 2            ; завантажити в регістр r16 константу $ 2
out DDRB, r16                   ; встановити лінію 2 порту B на висновок, всі інші
                        ; Лінії на введення
clr r16                         ; очистити регістр r16
out PORTB, r16                  ; встановити логічний нуль на всі лінії порту B
nop                             , нема операції (затримка для встановлення порту)
nop                             ;
ldi r16, $ 2            ; завантажити в регістр r16 константу $ 2
out PORTB, r16                  ; встановити логічну одиницю на лінію 2 порту B
nop                             , нема операції (затримка для встановлення порту)
nop                             ;
clr r16                         ; очистити регістр r16
out PORTB, r16                  ; встановити логічний нуль на всі лінії порту B
```

```
nop                    , нема операції (затримка для встановлення порту)
nop                    ;
```

Для очищення або встановлення однієї або декількох ліній порту безвтручання в роботу інших при використанні команд IN, OUT можна використовувати наступний спосіб:

```
in r16, DDRB           ; вважати в регістр r16 вміст регістра DDRB
ori r16, $ 2           ; встановити біт 2 в регістрі r16
out DDRB, r16          ; записати вміст r16 в DDRB
in r16, PORTB          ; зчитати в регістр r16 вміст регістра PORTB
andi r16, $ FD         ; очистити біт 2 в регістрі r16
out PORTB, r16         ; записати вміст r16 в PORTB
nop                    , нема операції (затримка для встановлення порту)
nop                    ;
in r16, PORTB          ; зчитати в регістр r16 вміст регістра PORTB
ori r16, $ 2           ; встановити біт 2 в регістрі r16
out PORTB, r16         ; записати вміст r16 в PORTB
nop                    , нема операції (затримка для встановлення порту)
nop                    ;
in r16, PORTB          ; зчитати в регістр r16 вміст регістра PORTB
andi r16, $ FD         ; очистити біт 2 в регістрі r16
out PORTB, r16         ; записати вміст r16 в PORTB
nop                    , нема операції (затримка для встановлення порту)
nop                    ;
```

Порядок виконання роботи

1. Ознайомитися з режимами роботи портів вводу-виводу мікроконтролерів серії AVR. Вивчити особливості функціонування та налаштуваннязовнішніх переривань. Ознайомитися зі структурою регістрів, використовуванихпри налаштуванні портів вводу-виводу.

2. Виписати індивідуальні початкові дані для виконання роботи (див. табл.).

Вариант	Имя порта	Ввод	Вывод	Подтяжка	Внешние прерывания
1	A	0,2	1,3	0,4,5	INT0 ⨯, INT1 ⨍, INT2 ⨿
2	B	1,3	2,4	0,1,5	INT0 ⨯, INT1 ⨍, INT2 ⨍
3	C	2,4	3,5	0,1,2	INT0 ⨯, INT1 ⨿, INT2 ⨿
4	D	3,5	4,6	0,1,3	INT0 ⨯, INT1 ⨿, INT2 ⨍
5	E	4,6	5,7	0,1,4	INT0 ⨯, INT1 ⨅, INT2 ⨿
6	A	5,7	0,6	1,2,5	INT0 ⨯, INT1 ⨅, INT2 ⨍
7	B	0,1,3	2	0,1,5	INT0 ⨍, INT1 ⨅, INT2 ⨿
8	C	1,2,4	3	0,1,5,6	INT0 ⨍, INT1 ⨍, INT2 ⨍
9	D	2,3,5	4	0,1,2,7	INT0 ⨍, INT1 ⨍, INT2 ⨿
10	E	3,4,6	5	0,3,4	INT0 ⨍, INT1 ⨿, INT2 ⨍
11	A	4,5,7	6	1,3,4	INT0 ⨍, INT1 ⨿, INT2 ⨿
12	B	5,6	7	2,3,4,5	INT0 ⨍, INT1 ⨅, INT2 ⨍
13	C	2	1,3,4	2,5,6	INT0 ⨍, INT1 ⨅, INT2 ⨿
14	D	3	2,4,5	3,7	INT0 ⨅, INT1 ⨍, INT2 ⨍
15	E	4	3,5,6	2,4,7	INT0 ⨅, INT1 ⨍, INT2 ⨿
16	A	5	4,6,7	1,3	INT0 ⨅, INT1 ⨿, INT2 ⨍
17	B	6	0,1,2	5,6	INT0 ⨅, INT1 ⨿, INT2 ⨿
18	C	7	0,2,3	5,7	INT0 ⨅, INT1 ⨅, INT2 ⨍
19	D	0,4	1,2,3	4,6,7	INT0 ⨅, INT1 ⨅, INT2 ⨿
20	E	1,5	2,3,4	5,6,7	INT0 ⨿, INT1 ⨍, INT2 ⨍
21	A	2,6	3,4,5	6,7	INT0 ⨿, INT1 ⨍, INT2 ⨿
22	B	3,7	4,5,6	1,3,7	INT0 ⨿, INT1 ⨿, INT2 ⨍
23	C	4,7	5,6	1,4,7	INT0 ⨿, INT1 ⨿, INT2 ⨿
24	D	1,3,6	2,4,5	1,3,6,7	INT0 ⨿, INT1 ⨅, INT2 ⨍
25	E	2,4,7	1,3,5,6	0,2,4,7	INT0 ⨿, INT1 ⨅, INT2 ⨿
⨿ - спадающий фронт, ⨍ - возрастающий фронт ⨅ - низкий логический уровень, ⨯ - смена логического уровня					

3. Запустити AVR Studio, на одному з доступних локальних дисківстворити проект з ім'ям "PORTS". Скласти програму ініціалізаціїпортів вводу-виводумікроконтролера ATmega8515 відповідно до свого індивідуального завдання. Програма повинна включати наступне:

- налаштування на введення зазначених ліній (колонка "Введення") порту з вказаним ім'ям (колонка "Ім'я порту");
- налаштування на вивід зазначених ліній (колонка "Висновок") порту ззазначеним ім'ям (колонка "Ім'я порту");
- підключенняпідтягуючих резисторів до вказаних ліній (колонка "Підтяжка") порту з вказаним ім'ям (колонка "Ім'я порту");
- налаштування переривань INT0, INT1, INT2 на відповідну чутливість (колонка "Зовнішні переривання");
- дозвіл переривань INT0, INT1, INT2;
- ініціалізація показника стека;
- установка глобального прапора переривання I.

4. Скомпілювати проект і, в разі необхідності, виправитипомилки.

5. За допомогою вбудованого симулятора провести покрокову відладку програми та переконатися в правильності її функціонування.

Контрольні питання

1. Керуючі регістри портів вводу-виводу: призначення, структура.
2. Конфігурація портів вводу-виводу. Підключення підтягуючих резисторів.
3. Зовнішні переривання. Налаштування чутливості переривання.
4. Асинхронні і синхронні типи переривань. Вивід мікроконтролера із сплячого режиму.
5. Регістри MCUCR, EMCUCR, GICR, GIFR.
6. Робота з портами вводу-виводу.

Необхідне програмне забезпечення та обладнання

1. AVR Studio v.4.11
2. Повний опис набору інструкцій мікроконтролерів серіїAVR "Instruction Set.pdf"

Перелік питань на підсумковий контроль

1. Вкажіть, що собою представляє мікроконтролер.
2. Вкажіть відмінність між мікропроцесором та мікроконтролером.
3. Вкажіть, що собою представляють поняття: RAM, ROM, EEPROM, I/O Ports.
4. Вкажіть, що собою представляють поняття: Serial I/O, Timer/Counter, PWM, ADC, DAC.
5. Опишіть типи МК та наведіть їх приклади.
6. Опишіть архітектуру доступу до пам’яті МК Фон-Неймана.
7. Опишіть Гарвардську архітектуру доступу до пам’яті МК.
8. Вкажіть, що собою представляє система команд CISC.
9. Вкажіть, що собою представляє система команд RISC.
10. Вкажіть, що собою представляє модель МК ATmega32A.
11. Опишіть структуру ядра AVR.
12. Опишіть арифметично-логічний пристрій (ALU) МК.
13. Вкажіть, як відбувається поділ регістрів загального призначення.
14. Вкажіть, що собою представляє регістер стану SREG.
15. Вкажіть, що собою представляє модуль переривань.
16. Вкажіть на які області поділений адресний простір SRAM.
17. Вкажіть, що собою представляють директиви .DSEG, .CSEG, .ESEG та .byte у МК.
18. Опишіть за допомогою яких директив відбувається розміщення масивів даних.
19. Опишіть директиви .org, .def, . equ.
20. Опишіть поняття макросів для програмування МК.
21. Опишіть набір основних функцій компілятора AVR Studio.
22. Вкажіть арифметичні, порозрядні, логічні та умовні операції компілятора AVR Studio.
23. Опишіть поняття абсолютного (прямого) переходу jmp.
24. Опишіть поняття відносного переходу rjmp.
25. Опишіть поняття непрямого переходу ijmp.
26. Вкажіть призначення наступних прапорців регістру стану SREG: C, Z, N.
27. Вкажіть призначення наступних прапорців регістру стану SREG: V, S, H.
28. Вкажіть призначення наступних прапорців регістру стану SREG: H, T, I.
29. Опишіть поняття команд умовного переходу у програмуванні МК.
30. Опишіть поняття команд переходу групи Branch у програмуванні МК.
31. Опишіть поняття команд переходу групи Compare у програмуванні МК.
32. Наведіть, як можна утворювати умовні конструкції на основі команди ср.

33. Наведіть приклад асемблерної реалізації умовної конструкції з умовою (r16<r17).
34. Опишіть поняття умовних команд групи Skip у програмуванні МК.
35. Опишіть поняття стеку у програмуванні МК.
36. Вкажіть, що собою представляють команди push та pop.
37. Опишіть поняття підпрограм у програмуванні МК.
38. Опишіть команди викликів з підпрогам: call, rcall та icall.
39. Вкажіть, яким чином реалізуються переривання у програмуванні МК.
40. Наведіть приклади векторів переривань для ATmega32A.
41. Наведіть роботу переривань на виходах МК (зовнішніх переривань).
42. Вкажіть, які регістри доступні для роботи зі зовнішніми перериваннями.
43. Опишіть відмінності макросів від підпрограм у програмуванні МК.
44. Наведіть приклад макросу, що виводить у будь-який регістр вводу/виводу константу.
45. Наведіть приклад макросу, що реалізує команду додавання до будь-якого регістра загального призначення константи.
46. Наведіть приклад макросу, що дає можливість присвоювати константу старшій та молодшій частині файлів регістрів загального призначення.
47. Опишіть поняття роботи з даними у SRAM.
48. Опишіть поняття роботи з даними у FLASH.
49. Опишіть, як відбувається кодування семисегментного індикатора зі спільним катодом.
50. Вкажіть, які регістри вводу/виводу використовують для роботи з EEPROM.
51. Вкажіть, що виконують команди: EERIE, EEMWE, EEWE, EERE.
52. Опишіть алгоритм запису одного байту в EEPROM-пам’ять.
53. Опишіть алгоритм читання одного байту в EEPROM-пам’ять.

Література та джерела

1. Бочаров С.Ю. Мікропроцесорна техніка. Навчальний посібник. – Рівне: НУВГП, 2006. – 163с.

2. Ю.І. Якименко, Т.О. Терещенко, Є.І. Сокол, В.Я. Жуйков, Ю.С. Петергеря. Мікропроцесорна техніка. 2-ге вид., переробл. та доповн. – К.: Політехніка, Кондор, 2004. – 440 с.

3. Терещенко Т.О. Мікропроцесорна техніка: Підручник. - К.: Видавництво „Політехнік”, 2003. – 440 с.

4. Павельчак А.Г., Самотий В.В., Яцук Ю.В. Програмування мікроконтролерів систем автоматики: конспект лекцій. – Львів: Львівська політехніка, 2012. – 143 с.

5. ATmega48A/PA/88A/PA/168A/PA/328/P DATASHEET – Atmel Corporation. – 657 с.

6. Локазюк В.М. Мікропроцесори та мікроЕОМ у виробничих системах: Посібник. Серія "Альма-матер". – Київ: Академія, 2002. – 367с.

7. Буняк А. Електроніка та мікросхемотехніка. Тернопіль, 2001 – 382 с.