НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет

імені Івана Пулюя

Кафедра автоматизації технологічних

процесів та виробництв

Методичні вказівки
до лабораторної роботи №9

« Виконання операцій з портами та таймерами МК ATmega32 на програмному симуляторі AVR Simulator IDE» з курсу «Проектування мікропроцесорних систем керування технологічними процесами»

Тернопіль 2020

Методичні вказівки до лабораторної роботи №9 Виконання операцій з портами та таймерами МК ATmega32 на програмному симуляторі AVR Simulator IDE з курсу «Проектування мікропроцесорних систем керування технологічними процесами».

Методичні вказівки розглянуті і схвалені кафедрою «Автоматизація технологічних процесів та виробництв», протокол № 8 від 18.02.2020 р.

Відповідальні за випуск доцент, к.т.н. Медвідь В.Р.,
асистент Пісьціо В.П.

Лабораторна робота №9

Виконання операцій з портами та таймерами МК ATmega32 на програмному симуляторі AVR Simulator IDE

1. Інтерфейс програмного симулятора AVR Simulator IDE

Основне вікно програми AVR Simulator IDE має вигляд, показаний на (рис. 1).

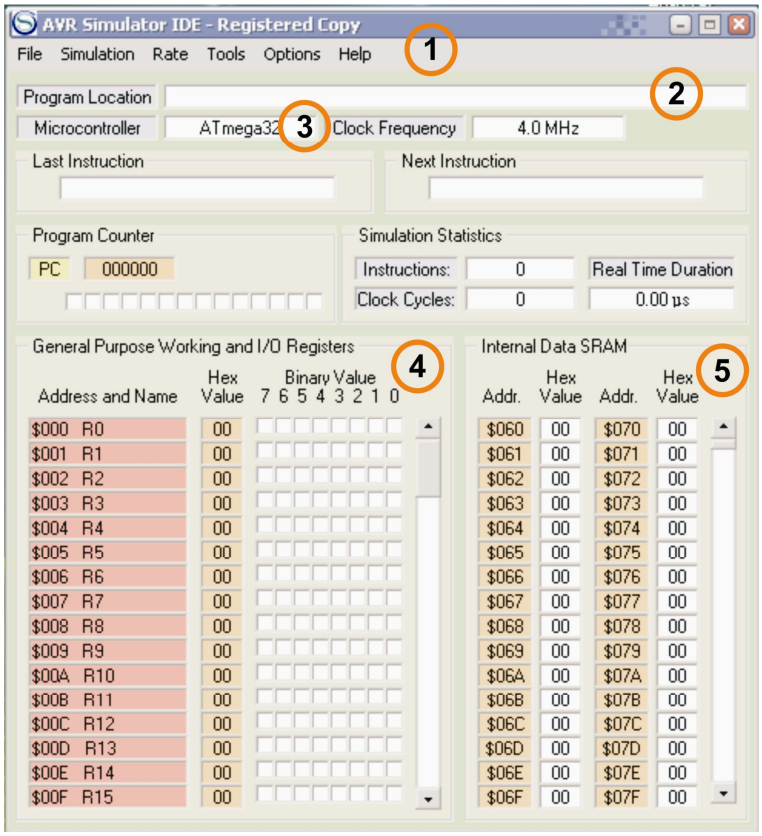

Рис. 1. Основне вікно програми AVR Simulator IDE

У верхній частині знаходяться меню, через які можна отримати доступ до основних і додаткових модулів програми (поз. 1)(рис. 1).

В рядку Program Location вказано шлях до обраної програми і її ім'я (поз. 2).

В рядку Microcontrollers, відображається тип обраного мікроконтролера (поз. 3).

У нижній частині вікна є дві панелі (поз.4 і поз.5), де відображається стан внутрішніх регістрів мікроконтролерів AVR (регістрів загального користування та регістрів вводу/виводу), та SRAM внутрішніх даних відповідно.

Також у основному вікні відображені лічильник програм, мнемоніка останньої виконуваної інструкції, мнемоніка наступної інструкції, що буде виконуватися, цикли та інструкції лічильника і тривалість імітації в режимі реального часу.

2. Послідовність роботи з програмним симулятором наступна:

- запуск програми AVR Simulator IDE;
- вибір типу мікроконтролера, для якого написана програма;
- вибір частоти кварцового генератора (впливає тільки на відображувані програмою дані про час виконання програми або команди, але не на швидкість роботи програми, що налагоджуються в AVR Simulator IDE);
- завантаження програми у вигляді HEX-файлу або запуск вбудованого компілятора мови асемблера і написання в ньому потрібної програми;
- вибір потрібних модулів віртуальних пристроїв;
- вибір швидкості і режиму роботи програми симулятора;
- запуск процесу симуляції роботи програми на обраному МК.

Якщо потрібно скористатися для роботи з симулятором власною програмою або внести зміни у вже розроблену, необхідно створити або завантажити для цього файл асемблера, з якого після компіляції буде створений необхідний для роботи з симулятором hex-файл.

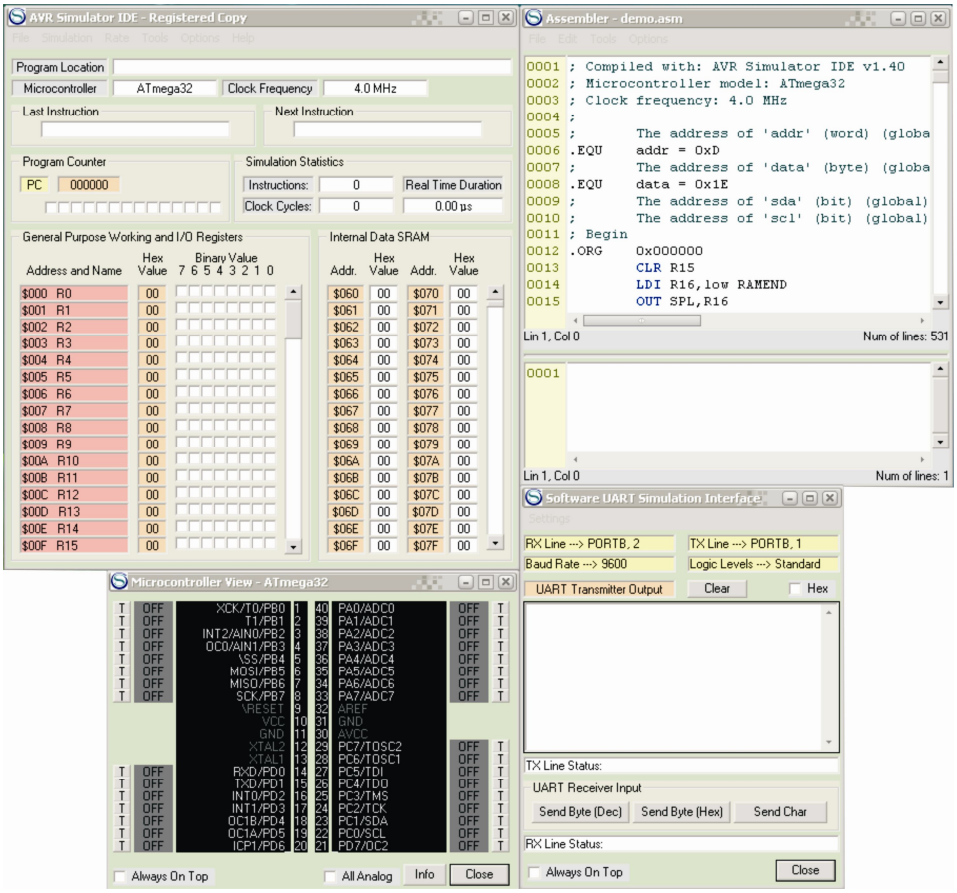

Рис. 2 Вікно симулятора з полем компілятора Assembler, апаратними виводами контролера, полем послідовного інтерфейсу

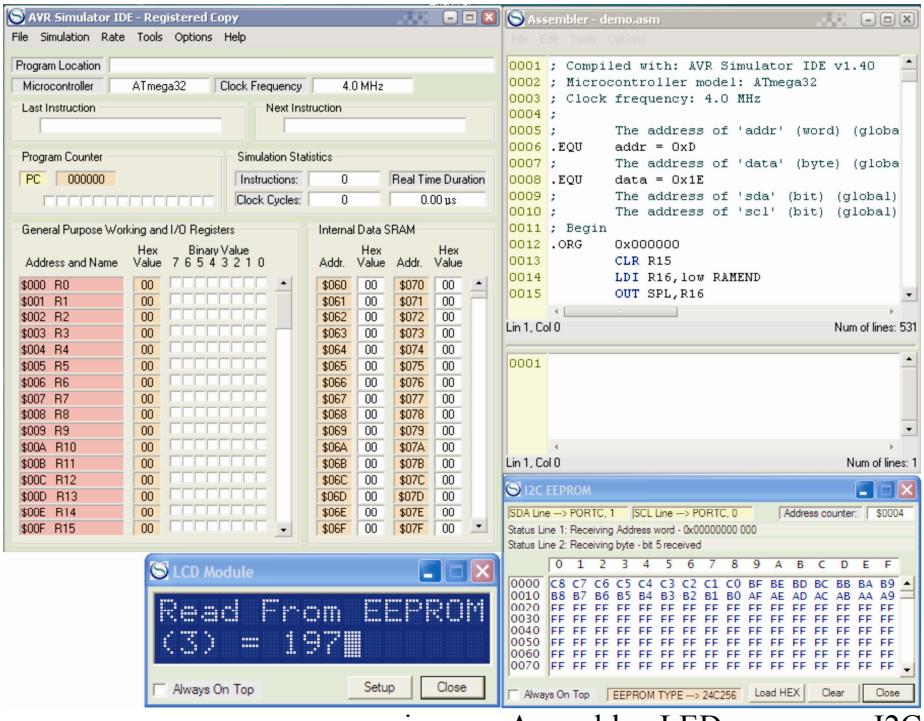

Рис. 3 Вигляд симулятора з полем компілятора Assembler, LED- модулем, I2C EEPROM

Для цього:

1. Натиснути Options | Assembler. Відкриється вікно компілятора Assembler – UNTITLED (рис. 2);

2. У вікні Assembler натиснути опцію File. Розкриється закладка, з якої для створення нового файлу потрібно натиснути New, а для завантаження вже створеного – OPEN.

3. Після вибору і завантаження файлу (з розширенням .asm), його текст з'явиться у вікні Assembler .

4. Для компіляції створеного або завантаженого і потім зміненого файлу, натисніть Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу і, одночасно, при відсутності помилок, буде створений одноіменний hex-файл.

3. Послідовність роботи з симулятором при виконанні програм

Виконати програму відповідно до вказаного викладачем завдання в AVR Simulator ID, для чого необхідно:

1. Запустити AVR Simulator IDE;

2. Натиснути Options | Select Microcontroller;

3. Вибрати ATmega32 і натиснути кнопку Select;

4. Натиснути Tools і у вікні, що розкриється, вибрати «Assembler». Відкриється вікно компілятора «Assembler – UNTITLED» (рис. 2);

5. Набрати текст заданої програми у вікні «Assembler»;

6. Натиснути Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу;

7. Одночасно, при відсутності помилок, буде створений файл з розширенням «hex», для якого можна вибрати ім'я та шлях для запису. Записати його на «Робочий стіл» комп'ютера;

8. Вибрати File | Load Program і завантажити створений файл hex-файл;

9. В основному вікні симулятора натиснути Rate | Step By Step, а далі вибрати опцію Simulation і натиснути Start. Симулятор готовий до виконання програми в кроковому режимі;

10. Для виконання наступної команди програми потрібно натиснути на закладку STEP, яка з'явиться справа від закладки HELP вгорі основного вікна симулятора після вибору крокового режиму його роботи;

11. Для виконання програми в автоматичному режимі потрібно вибрати Rate | Extremely Fast simulation rate;

12. Щоб зупинити виконання програми, потрібно натиснути Simulation | Stop.

Вміст регістрів контролера, які використовуються при виконанні команд програми, знайти в області регістрів Adress and Name, яка розташована в лівій нижній частині основного вікна симулятора (виділені рожевим кольором). Всі регістри восьмирозрядні.

В процесі виконання програми по зміні кольору комірок видно, вміст яких регістрів змінюється. Забарвлення комірки відповідного розряду регістру помаранчевим кольором означає наявність "1", білим - "0".

4. Завдання на лабораторну роботу

4.1 **Завдання 1**

Для схеми (рис. 4) реалізувати наступну задачу:

При натисканні кнопки S1, яка під'єднана до лінії PA0 порту PA, на лінії порту PB0…PB7, до яких під'єднані світлодіоди, вивести значення, що дорівнює кількості натискань кнопки, у бінарному коді. Одночасно записати це значення у регістр загального призначення МК.

Послідовність виконання завдання:

1. В опції «Tools» симулятора вибрати поля «Microcontroller View» та «8 LED Board» (рис. 5).

2. Виконати програму в автоматичному режимі, для чого вибрати «Rate | Fast».

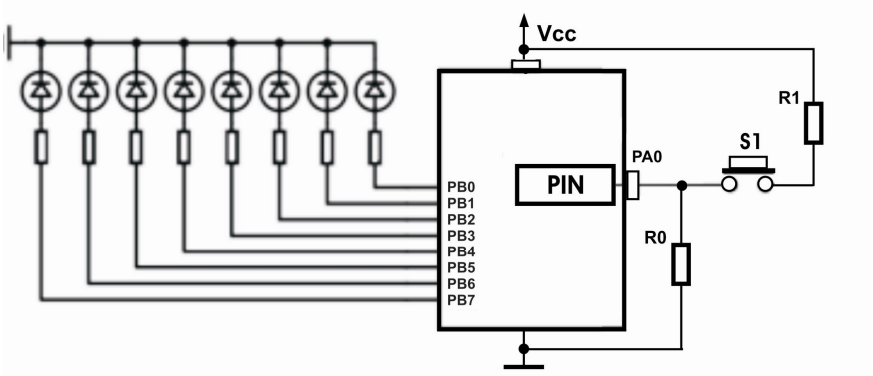

Рис. 4 Принципова схема до Завдання 1

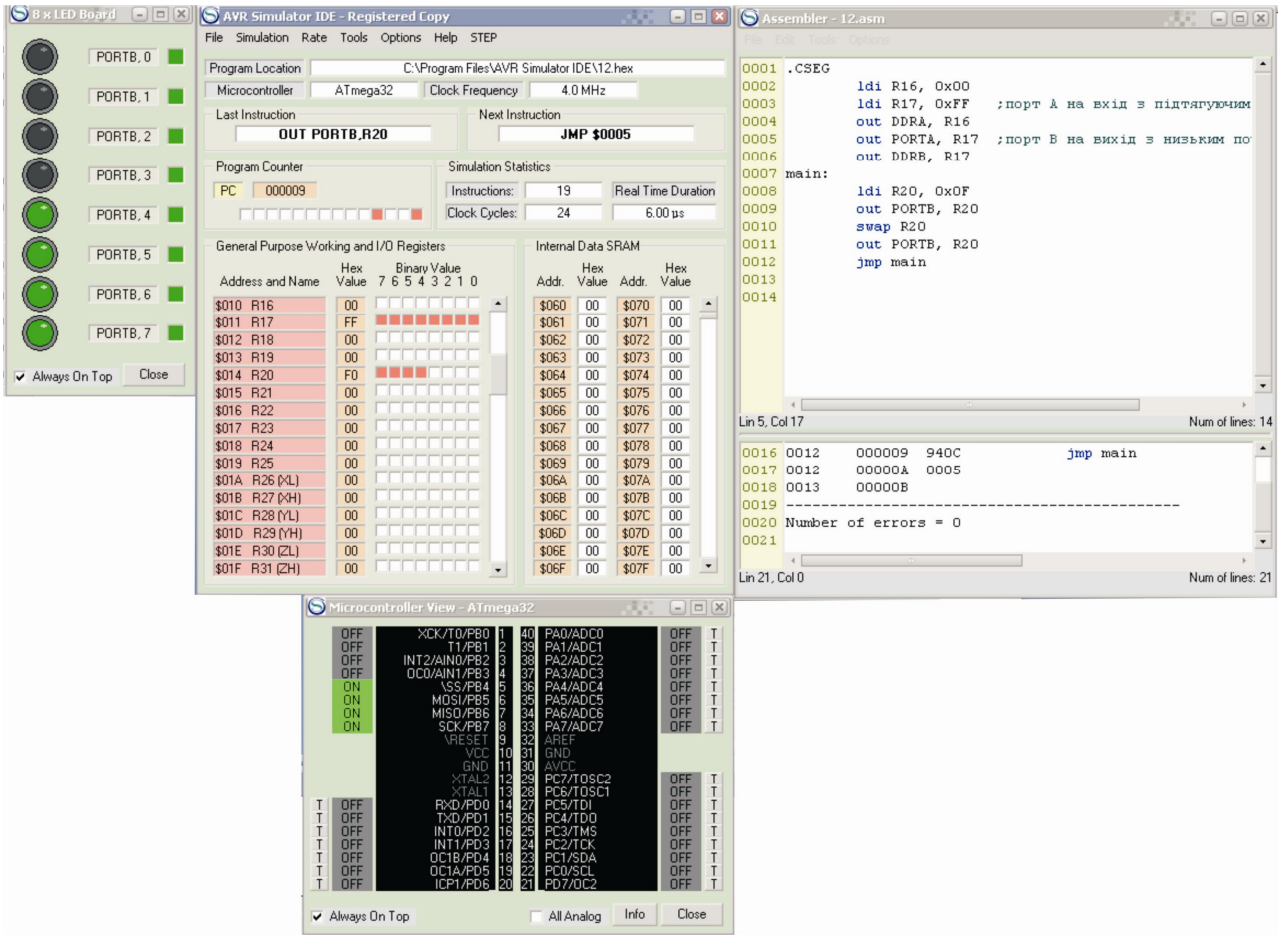

Рис. 5 Інтерфейс симулятора з полем «Assembler», панелями «8 x LED Board» та «Microcontroller Viev»

3. Після запуску програми на виконання (вибрати опцію Simulation і натиснути Start) встановити курсором на лінії PA0 логічну «1» . При цьому значення «OFF» на лінії зміниться на «ON» і зафарбується зеленим кольором. На виході PB0 порту PB засвітиться діод.

4. Після того, як в процесі виконання програми лінія PA0 знову стане неактивною (перейде в стан «OFF»), знову курсором подати на неї активний рівень (перевести її в стан «ON»). Такими діями імітується робота кнопки S1.

5. Повторити дії, вказані у пункті 4. Переконатися, що на виході порту PB буде з'являтися бінарний код, який відповідає кількості натискань кнопки.

Програма для виконання:

```
;основна програма
.CSEG
    ldi R16, 0x00
    ldi R17, 0xFF
;Порт B на вихід з низьким початковим рівнем
    out DDRB,R17
    out PORTB,R16

M1:
    out DDRA, R16          ; запрограмувати порт A на ввід
    mov R21,R20

main:
    ldi R18,0x10           ; завантажити лічильник циклів
M2:
    in R20,PINA            ;записати логічний рівень на лініях порту A в регістр R20"
    dec R18                ; ввести часову затримку при опитуванні лінії
    brne M2
    sbis PINA,0            ; якщо на лінії A0 порту A «1», то пропустити наступну
                           ; команду
    jmp main
    add R20,R21
    out PORTB,R20          ;вивести вміст регістра R20 в порт B
    out DDRA,R17           ; запрограмувати лінії порту A на вивід
    cbi PORTA,0            ; скинути лінію A0 порту A в нуль
    jmp M1
```

За прикладом даної програми виконати наступне:

1. Розробити програму відповідно до вказаного варіанту: 1) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PA; 2) кнопка S1 – на вхід лінії PA0, вивід на світлодіоди – через лінії порту PC; 3) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PC, 4) кнопка S1 – на вхід лінії PA4, вивід на світлодіоди – через лінії порту PB.

2. Роздрукувати текст програми з коментарями до кожної команди.

Завдання 2

Реалізувати керування двигуном постійного струму за допомогою ШІМ-сигналу у режимі Fast PWM з виводу OC0 таймера T0. ШІМ-сигнал дискретно розбити на 8 рівнів, кожен з яких виводиться окремо на світлодіоди через порт A.

Збільшення чи зменшення ширини імпульсу виконувати за допомогою 2-х кнопок, підключених до ліній PB6 (UP-збільшити) та PB7 (DOWN- зменшити) порту B.

Керування двигуном виконується за допомогою транзисторного ключа (рис. 6) зміною щільності імпульсів з виходу OCO таймера, які подаються на базу транзистора.

Для захисту транзисторного ключа від перенапруги паралельно з двигуном увімкнено діод.

Для імітації роботи кнопки спочатку натиснути курсором на відповідному вході мікроконтролера на панелі «Microcontroller Viev», до якого під'єднана кнпка (перевести лінію в стан «ON»), та натиснути знову (перевести лінію в стан «OFF»).

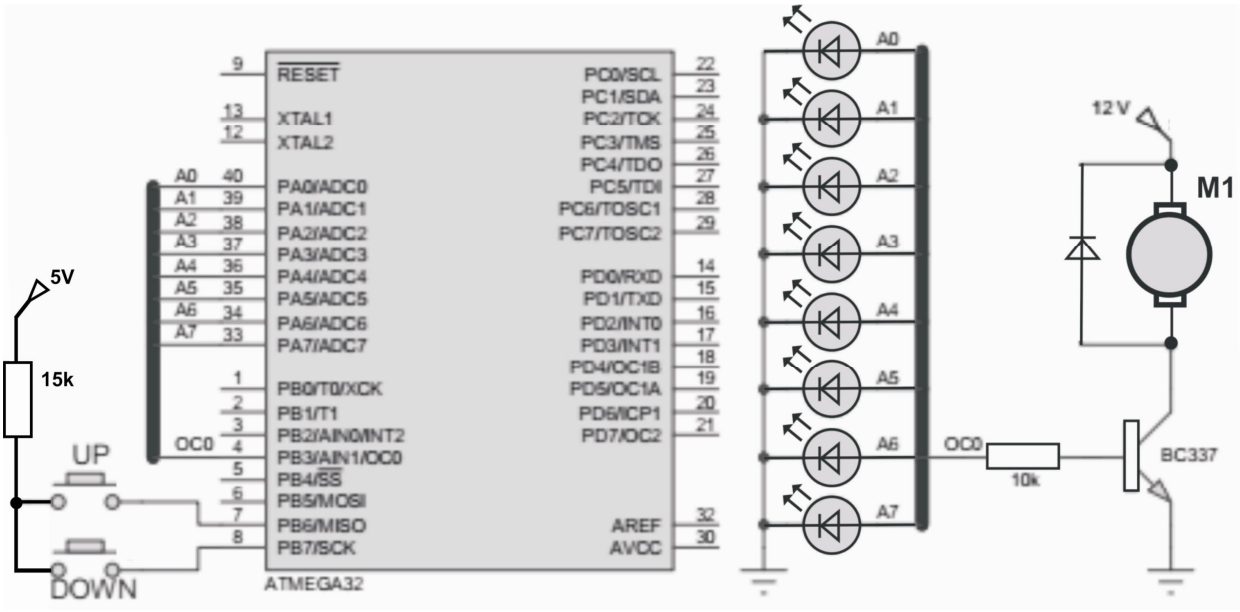

Рис. 6 Принципова схема до Завдання 2

При виконанні програми на симуляторі до виводу OC0 мікроконтролера замість транзисторного ключа, який апаратно не реалізується симулятором, підключити осцилоскоп, вибравши його в опції «Tools» (Рис. 7).

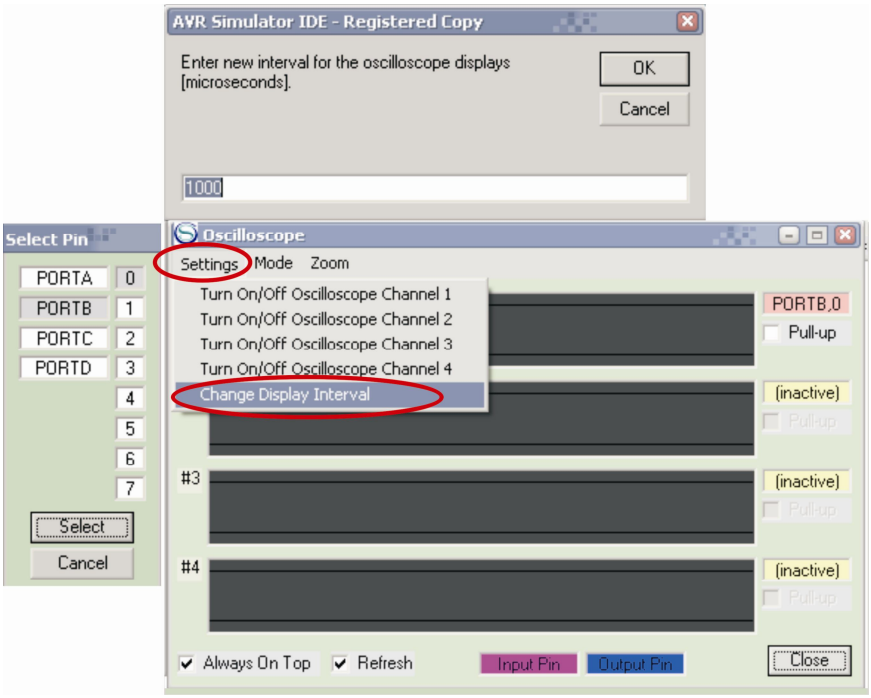

Рис. 7 Панель симулятора «Oscilloscope»

За його допомогою визначити, як буде змінюватися щільність імпульсів в залежності від впливу кнопок «UP» та «DOWN».

Для налаштування панелі необхідно:

- вибрати порт і номер лінії порту, до якого буде під'єднано осцилоскоп, для чого натиснути курсором на панелі «Select Pin» (зліва на рис. 7) спочатку на назві порту, далі на номері лінії, після чого натиснути кнопку «Select»;
- зайти в опцію «Setting» і вибрати закладку «Change Display Interval». Розкриється вікно (вгорі на рис. 7), в якому вкажіть значення для інтервалу, що дорівнює 10 000 мкс.

На рис. 8 показано формування сигналу ШІМ PWM на виводі OCn (вивід PB3 порту PB) з використанням регістрів TCNTn та OCRn (для таймера T0 відповідно вивід OC0, регістри TCNT0, OCR0).

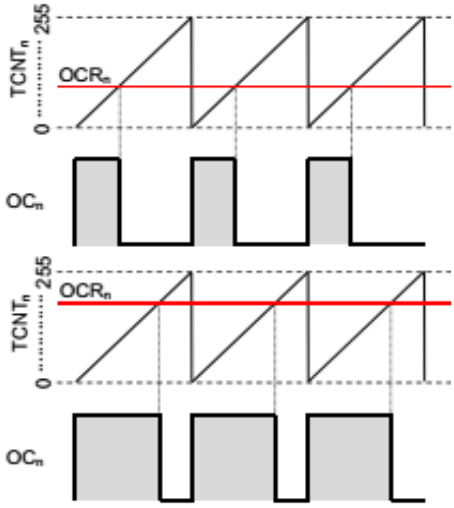

Рис. 8 Формування ШІМ-сигналу у режимі Fast PWM

Для виконання завдання необхідно розглянути призначення регістрів 8-розрядного таймера-лічильника T0 в режимі ШІМ для його програмування в цьому режимі.

Таймер T0 містить три регістри:

- рахунковий регістр **TCNT0**,
- регістр порівняння **OCR0**,
- конфігураційний регістр **TCCR0**.

Крім того, є ще три регістри, що відносяться до всіх трьох таймерів ATmega32:

- конфігураційний регістр **TIMSK**,
- статусний регістр **TIFR**,
- регістр спеціальних функцій **SFIOR**.

Регістр TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Це 8-розрядний рахунковий регістр. Коли таймер працює, по кожному імпульсу тактового сигналу значення TCNT0 змінюється на одиницю. В залежності від режиму роботи таймера, рахунковий регістр може або збільшуватися, або зменшуватися.

Регістр TCNT0 можна як зчитувати, так і записувати. Останнє використовується, коли потрібно задати його початкове значення.

В процесі роботи таймера змінювати вміст TCNT0 не рекомендується, так як це блокує схему порівняння на один такт.

Регістр OCR0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Це 8-розрядний регістр порівняння. Його значення постійно порівнюється з вмістом рахункового регістру TCNT0, і в разі збігу таймер може виконувати певні дії - викликати переривання, змінювати стан виводу OC0 і т.д. в залежності від режиму роботи.

Значення з OCR0 можна як зчитувати, так і записувати.

Регістр TCCR0 (Timer/Counter Control Register)

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Це конфігураційний регістр таймера-лічильника T0, він визначає джерело тактування таймера, коефіцієнт передподільника, режим роботи таймера-лічильника T0 і поведінку виводу OC0. По суті, це найважливіший регістр.

Біти **CS02, CS01, CS00** (Clock Select) - визначають джерело тактової частоти для таймера T0 і задають коефіцієнт передподільника. Всі можливі стани цих бітів описані в таблиці нижче.

CS02	CS01	CS00	Призначення
0	0	0	Джерела тактування немає. Таймер зупинено.
0	0	1	Тактова частота МК.
0	1	0	Тактова частота МК/8.
0	1	1	Тактова частота МК/64.
1	0	0	Тактова частота МК/256.
1	0	1	Тактова частота МК/1024.
1	1	0	Зовнішнє джерело на виводі T0. Спрацювання по задньому фронту
1	1	1	Зовнішнє джерело на виводі T0. Спрацювання по передньому фронту

Таким чином, таймер-лічильник може бути зупинений, може тактуватися від внутрішнього генератора, а також від сигналу на лінії T0.

Біти **WGM10, WGM00** (Wave Generator Mode) - визначають режим роботи таймера-лічильника T0.

Всього їх може бути чотири: нормальний режим (normal), скидання таймера при збігу (CTC), і два режими широтно-імпульсної модуляції (FastPWM і Phase Correct PWM).

Всі можливі значення бітів описані в таблиці нижче.

WGM01	WGM00	Режим роботи таймера-лічильника
0	0	Normal
0	1	PWM, Phase Correct
1	0	CTC
1	1	Fast PWM

Біти **COM01, COM00** (Compare Match Output Mode) визначають поведінку виводу OC0.

Якщо хоч один з цих бітів встановлений в «1», то вивід OC0 перестає функціонувати як звичайний вивід загального призначення і підключається до схеми порівняння таймера лічильника T0.

Однак, при цьому він повинен бути ще налаштований на вихід.

COM01	COM00	Призначення
0	0	Таймер-лічильник відключений від виводу OC0
0	1	Таймер-лічильник відключений від виводу OC0
1	0	Скидається в 0 при рівності вмісту регістрів TCNT0 та OCR0, встановлюється в 1 при скиданні лічильника - в режимі Fast PWM, або при співпадінні при зворотній лічбі - в режимі Phase Correct PWM
1	1	Встановлюється в 1 при рівності вмісту регістрів TCNT0 та OCR0, скидається в 1 при скиданні лічильника - в режимі Fast PWM, або при співпадінні при зворотній лічбі - в режимі Phase Correct PWM

Поведінка виводу OC0 залежить від режиму роботи таймера-лічильника T0.

Останній біт регістра TCCR0 - біт **FOC0** (Force Output Compare). Цей біт призначений для примусової зміни стану виводу OC0. Він встановлюється в одиницю тільки для режимів Normal і CTC. В інших режимах біт **FOC0** має бути скинутий в нуль, а стан виводу OC0 змінюється відповідно до значень бітів **COM01, COM00**.

Біт **FOC0** не викликає переривання і не скидає таймер в CTC режимі.

Таким чином, для вибору режиму роботи таймера від внутрішнього генератора, з коефіцієнтом передподільника, рівним 1, в режимі ШІМ Fast PWM, в регістр TCCR0 потрібно записати наступне значення - **01111001b**.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	0	**1**	**1**	**1**	**1**	**0**	**0**	**1**	

Регістр TIMSK (Timer/Counter Interrupt Mask Register)

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Спільний регістр для всіх трьох таймерів ATmega32, він містить прапорці дозволу переривань. Таймер T0 може викликати переривання при переповненні рахункового регістра **TCNT0** і при збігу вмісту рахункового регістра з вмістом регістра порівняння **OCR0.**

Відповідно, для таймера T0 в регістрі TIMSK зарезервовані два біти - це **TOIE0** і **OCIE0**. Решта бітів відносяться до інших таймерів.

TOIE0 –значення біту «**0**» забороняє переривання за подією переповнення, «**1**» - дозволяє.

OCIE0 –значення **«0»** забороняє переривання за подією збіг, а **«1»** - дозволяє.

Переривання будуть викликатися, тільки якщо встановлено біт глобального дозволу переривань - біт **«I»** регістра **SREG**.

Регістр TIFR (Timer/Counter0 Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Спільний для всіх трьох таймерів-лічильників регістр. Містить статусні прапорці, які встановлюються при виникненні подій. Для таймера T0 - це переповнення рахункового регістра **TCNT0** і збіг рахункового регістра з регістром порівняння **OCR0**.

Якщо в ці моменти в регістрі TIMSK дозволені переривання і встановлений біт I, то мікроконтролер викличе відповідну підпрограму обробки переривання.

Прапорці автоматично очищаються при запуску підпрограми переривання. Також це можна зробити програмно, записавши «1» у відповідний прапорець.

TOV0 - встановлюється в 1 при переповненні рахункового регістра.

OCF0 - встановлюється в 1 при збігу рахункового регістра з регістром порівняння

Регістр SFIOR (Special Function IO Register)

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Один з його розрядів скидає 10-розрядний двійковий лічильник, який ділить вхідну частоту для таймера T0 і таймера T1.

Скидання здійснюється при встановленні біту **PSR10** (Prescaler Reset Timer / Counter1 і Timer / Counter0) в одиницю.

Програма для виконання

```
; основна програма
; імена для регістрів загального призначення
.def _temp1 =r16
.def _temp2 =r17
.def _temp3 =r18
.def _power =r19                    ; значення для виводу в OCR0
.def _leds =r20                     ; значення індикації світлодіодів
.def _val32 =r21                    ; 256\8=32 - значення дискретного рівня ШІМ-
                                    ; сигналу (256 - мах значення лічильника, 8 –
                                    ; кількість рівнів)
.CSEG
        ldi r24,0x30                ; завантаження лічильника циклів для часової
                                    ; затримки при опитуванні кнопок
        ldi r22,0x80
        ldi r23,0x02                ; код для ініціалізації регістра TIMSK
        out SREG,r22                ; дозвіл глобального переривання, прапорець «I»=1
            ; ініціалізація стеку (RAMEND – максимальна адреса РПД)
    ldi _temp1, Low(RAMEND)
    out SPL, _temp1
    ldi _temp1, High(RAMEND)
    out SPH, _temp1
            ; ініціалізація портів вводу/виводу
    ldi _temp1, 0x00
    ldi _temp2, 0xFF
            ; програмування порту A на вихід
    out DDRA, _temp2
    out PORTA, _temp1
    ldi _temp1, 0x0F
    ldi _temp2, 0xF0
            ; порт B – 0..3 вивід на вихід, 4..7 - на вхід
    out DDRB, _temp1
    out PORTB, _temp2
```

```
                ; ініціалізація таймера T0 -- fast PWM; fPWM = 31 250 Гц (коеф. передподільника
                ; рівний 1)
        ldi _temp1, 0b01111001
        out TCCR0, _temp1                   ; режим роботи ШІМ
        out TIMSK,r23                       ; дозвіл переривання при рівності  вмісту
                                            ; рахункового регістра з вмістом регістра OCR0
        clr _power
        out OCR0, _power                    ; вміст регістра OCR0 = 0
        clr _leds                           ;очистити регістр  leds = 0
        ldi _val32, 32                      ; val32 = 32
main:
B6:                     ; опитування кнопки UP
        sbis PINB, 6                        ; якщо натиснута кнопка UP («1» на вході PB0.6),
                                            ; пропустити наступну команду
        rjmp B7
        add _power, _val32                  ; power =power + 32
        brcc B6a                            ; якщо power <= 255
        ldi _power, 255                     ; power = 255
        B6a:
        out OCR0, _power                    ; OCR0=power
        lsl _leds                           ; зсув вмісту регістра leds вліво
        set                                 ; встановлення прапорця «T»=1 в регістрі SREG
        bld _leds,0                         ; завантаження вмісту «T» в нульовий біт leds
        out PORTA, _leds                    ; PORTA← leds
        rcall Pause                         ; виклик підпрограми затримки в часі
B7:                     ; опитування кнопки DOWN
        sbis PINB, 7                        ; якщо натиснута кнопка DOWN («1» на вході PB0.7),
                                            ; пропустити наступну команду
        rjmp end
        sub _power, _val32                  ; power =power – 32
        brcc B7a                            ; якщо power >= 0
        ldi _power, 0                       ; power = 0
B7a:
        out OCR0, _power                    ; OCR0=power
        lsr _leds                           ; зсув вмісту регістра вправо
        out PORTA, _leds                    ; PORTA← leds
        rcall Pause                         ; виклик підпрограми затримки
end:
        rjmp main
           ; підпрограма затримки
Pause:
        dec r24                             ; декремент регістра r24
        brne Pause                          ; перехід за міткою, якщо не нуль
        ret
```

Для роботи кнопок використовується програмна затримка Pause.

Для виконання програми вибрати в закладці «Rate» режим «Extremely Fast» .

За прикладом даної програми виконати наступне:

1. Розробити програму відповідно до вказаного варіанту: 1) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PA; 2) кнопка S1 – на вхід лінії PA0, вивід на

світлодіоди – через лінії порту PC; 3) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PC, 4) кнопка S1 – на вхід лінії PA4, вивід на світлодіоди – через лінії порту PB.

2. Роздрукувати текст програми з коментарями до кожної команди.

Завдання 3

Реалізувати керування двигуном постійного струму за допомогою ШІМ-сигналу, використовуючи в якості таймера регістр загального призначення R24 та регістра порівняння – регістр R19. Вивести ШІМ-модульований сигнал через лінію PB3 порту B.

ШІМ-сигнал дискретно розбитий на 8 рівнів, кожен з яких виводиться окремо на світлодіоди через порт B.

Збільшення чи зменшення швидкості обертання двигуна виконати за допомогою 2-х кнопок, підключених до ліній PB6 (UP) та PB7 (DOWN) порту B.

Для імітації роботи кнопки спочатку натиснути курсором на відповідному вході мікроконтролера на панелі «Microcontroller Viev», до якого під'єднана кнопка (перевести лінію в стан «ON»), та натиснути знову (перевести лінію в стан «OFF»).

При виконанні програми на симуляторі до виводу порту B (PB0) мікроконтролера замість транзисторного ключа, який апаратно не реалізується симулятором, підключити осцилоскоп, вибравши його в опції «Tools» (Рис. 7).

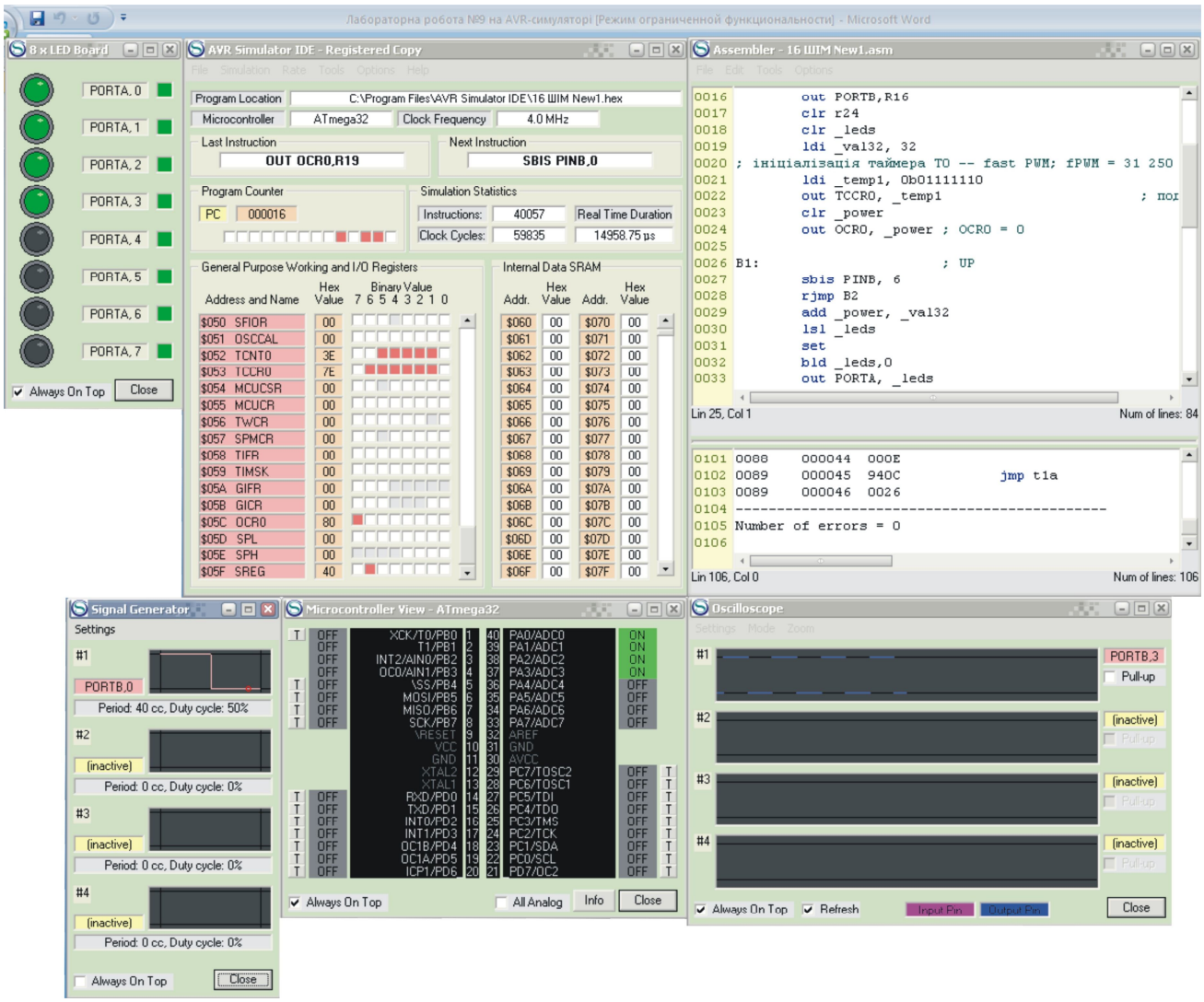

Рис. 9 Вікно симулятора з полем генератора «Signal Generator»

Для реалізації алгоритму роботи в режимі ШІМ PWM при тактуванні від зовнішнього генератора по входу таймера T0, використаємо один із каналів чотириканального генератора, що входить до складу симулятора – «Signal Generator» (рис. 9).

Для налаштування генератора необхідно:

- вибрати порт і номер лінії порту (вибрати PB3), до якого буде під'єднано генератор, для чого натиснути курсором у вікні «inactive» вибраного каналу (наприклад, каналу #1), після чого натиснути кнопку «Select»;
- натиснути курсором на вікні, що знаходиться під попереднім, і вибрати спочатку період розгортки генератора (ввести 40), а потім - тривалість імпульсу (вибрати 50%).

Для виконання програми вибрати в закладці «Rate» режим «Extremely Fast» .

Приклад програми для виконання:

```
; основна програма
; імена для регістрів загального призначення
.def _temp1 =r16
.def _temp2 =r17
.def _temp3 =r18
.def _power =r19                    ; значення для виводу в OCR0
.def _leds =r20                     ; значення індикації світлодіодів
.def _val32 =r21                    ; 256\8=32 - значення дискретного рівня ШІМ-
                                    ; сигналу (256 - мах значення лічильника, 8 –
                                    ; кількість рівнів)
.CSEG
        ldi R16, 0x00
        ldi R17, 0xFF
        ldi R25,0x0E
        out DDRA,R17                ; лінії 1..3 PORTB – на вихід, лінії 1,4..7- на вхід
        out PORTA,R16
        out DDRB, R25
        out PORTB,R16
        clr r24                     ; значення для виводу в TCNT0
        clr _leds
        ldi _val32, 32
        clr _power
        out OCR0, _power            ; OCR0 = 0
B1:             ; опитування стану кнопки «UP»
        sbis PINB, 6                ; якщо «1», то пропустити команду
        rjmp B2
        add _power, _val32
        lsl _leds
        set
        bld _leds,0
        out PORTA, _leds
t1:
        out OCR0, _power
        sbis PINB, 0                ; якщо «1» на лінії «PB0», то пропустити команду
        jmp t1
        inc r24
        out TCNT0, r24              ; завантаження лічильника
        cp _power,r24               ; порівняти вміст NCNT0 та
```

```
        breq m3 ; якщо дорівнює, то перехід за міткою
        cpi r24,0
        brne t1
        jmp m4

B2:                     ; опитування стану кнопки «DOWN»
        sbis PINB, 7
        rjmp B1
        sub _power,_val32
        lsr _leds
        out PORTA, _leds
t1a:
        out OCR0, _power
        sbis PINB, 0
        jmp t1a
        inc r24
        out TCNT0, r24
        cp _power,r24
        breq m3a
        cpi r24,0
        brne t1a
        jmp m4a
end:
        rjmp b1

M3:
        sbi portb,3
        Jmp t1
M4:
        cbi portb,3
        sbis PINB, 6
        sbic PINB, 7
        jmp b2
        jmp t1
M3a:
        sbi portb,3                     ; встановити лінію PB3 в «1»
        Jmp t1a
M4a:
        cbi portb,3                     ; скинути лінію PB3 в «0»
        sbis PINB, 7
        sbic PINB, 6
        jmp b1
        jmp t1a
```

Примітка:

Після того того, як за допомогою кнопок вибирається необхідний рівень напруги для порівняння, його значення буде зберігатися при роботі симулятора після вимкнення кнопки, а, отже, і значення відповідної ширини імпульсу на виході зберігається протягом часу, поки не відбудеться повторне натиснення якоїсь із кнопок.

За прикладом даної програми виконати наступне:

1. Розробити програму відповідно до вказаного варіанту: 1) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PA; 2) кнопка S1 – на вхід лінії PA0, вивід на світлодіоди – через лінії порту PC; 3) кнопка S1 – на вхід лінії PB0, вивід на світлодіоди – через лінії порту PC, 4) кнопка S1 – на вхід лінії PA4, вивід на світлодіоди – через лінії порту PB.
2. Роздрукувати текст програми з коментарями до кожної команди.
3. Роздрукувати копію екрану з виглядом стимулятора з панелями відповідно до рис. 9 після виконання програми.

Контрольні запитання

1. Використання AVR-мікроконтролерів.
2. Програмування портів мікроконтролера.
3. Організація циклів в роботі мікроконтролера.
4. Формат та використання регістрів загального призначення.
5. Призначення та позначення основних елементів програмної моделі мікроконтролера.

Література

1. 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.
2. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL, – [5-е изд., стер.] / Евстифеев А. В. – М.: Издательский дом «Додэка-XXI», 2008. 560 с.
3. Учебный курс AVR. Таймер - счетчик T0. Регистры. [Електронний ресурс]. Режим доступу: https://chipenable.ru/index.php/programming-avr/171-avr-timer-t0-ch1.html/

Додаток 1
Система команд мікроконтролерів AVR

Система команд AVR мікроконтролерів включає команди арифметичних і логічних операцій, команди передачі даних, команди, що керують послідовністю виконання програми і команди операцій з бітами.

Для зручності написання й аналізу програм всім операціям із системи команд крім двійкового коду зіставлені мнемокоди Ассемблера (символічні позначення операцій), що використовуються при створенні вихідного тексту програми.

Спеціальні програми-транслятори переводять потім символічні позначення в двійкові коди.

Спеціальна директива ассемблера **.device** забезпечує контроль відповідності команд, використовуваних у тексті програми, типу зазначеного процесора.

Під час виконання арифметичних, логічних чи операцій роботи з бітами ALU формує ознаки результату операції, тобто встановлює чи скидає біти в регістрі стану **SREG** (Status Register).

Регістр статусу - SREG - розміщений у просторі I/O за адресою $3F ($5F).

Таблиця 1 – Регістр статусу - SREG

Біти	7	6	5	4	3	2	1	0	
$3F ($5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**REG**
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початковий стан	0	0	0	0	0	0	0	0	

Bit 7 - I: Global Interrupt Enable - Дозвіл глобального переривання. Біт дозволу глобального переривання для дозволу переривання повинний бути встановлений у стан 1. Керування дозволом конкретного переривання виконується регістрами маски переривання GIMSK і TIMSK. Якщо біт глобального переривання очищений (у стані 0), то жодне з дозволів конкретних переривань, встановлених у регістрах GIMSK і TIMSK, не діє.

Біт I апаратно очищається після переривання і встановлюється для наступного дозволу глобального переривання командою RETI.

Bit 6 - T: Bit Copy Storage - Біт збереження копії. Команди копіювання біта BLD (Bit Load) і BST (Bit STore) використовують біт T, як біт джерело і біт призначення при операціях з бітами. Командою BST біт регістра копіюється до біту T, командою BLD біт T копіюється до регістру.

Bit 5 - H: Half Carry Flag - Прапор напівпереносу. Прапор напівпереносу вказує на напівперенос у ряді арифметичних операцій.

Bit 4 - S: Sign Bit, S = N V - Біт знаку. Біт S завжди знаходиться в стані, обумовленому логічною функцію АБО (OR) між прапором негативного значення N і доповненням до двох прапора переповнення V.

Bit 3 - V: Two's Complement Overflow Flag. Доповнення до двох прапора переповнення. Доповнення до двох прапора V підтримує арифметику доповнення до двох.

Bit 2 - N: Negative Flag – Прапор негативного значення. Прапор негативного значення N вказує на негативний результат ряду арифметичних і логічних операцій.

Bit 1 - Z: Zero Flag – Прапор нульового значення. Прапор нульового значення Z вказує на нульовий результат ряду арифметичних і логічних операцій.

Bit 0 - C: Carry Flag – Прапор переносу. Ознаки результату операції можуть бути використані в програмі для виконання подальших арифметично-логічних операцій чи команд умовних переходів.

Арифметичні і логічні конструкції

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
ADD	Rd,Rr	Підсумовування без переносу	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Підсумовування з переносом	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Вирахування без переносу	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Вирахування константи	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Вирахування з переносом	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Вирахування константи з переносом	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логічне И	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне И с константою	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 константою	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логічне що виключає АБО	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Побітна Інверсія	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Зміна знака (Доп. код)	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Установити біт (біти) у регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Скинути біт (біти) у регістрі	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Інкрементувати значення регістра	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декрементувати значення регістра	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Перевірка на нуль або заперечність	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Очистити регістр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Установити регістр	Rd = $FF	None	1
ADIW	Rdl,K6	Скласти константу і слово	Rdh:Rdl=Rdh:Rdl+ K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Вичитати константу зі слова	Rdh:Rdl=Rdh:Rdl - K 6	Z,C,N,V,S	2

Інструкції розгалуження

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
RJMP	k	Відносний перехід	PC = PC + k +1	None	2
IJMP	Немає	Непрямий перехід на (Z)	PC = Z	None	2
EIJMP	Немає	Розширений непрямий перехід на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Перехід	PC = k	None	3
RCALL	k	Відносний виклик підпрограми	STACK=PC+1, PC=PC + k+ 1	None	3/4*
ICALL	Немає	Непрямий виклик (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	Немає	Розширений непрямий виклик (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
RET	Немає	Повернення з підпрограми	PC = STACK	None	4/5*
RETI	Немає	Повернення з переривання	PC = STACK	I	4/5*
CPSE	Rd,Rr	Порівняти, пропустити якщо рівні	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Порівняти	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Порівняти з переносом	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Порівняти з константою	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Пропустити якщо біт у регістрі очищений	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3

SBRS	Rr,b	Пропустити якщо біт у регістрі встановлений	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Пропустити якщо біт у порту очищений	if(I/O(P,b)==0) PC=PC + 2 or 3	None	1/2/3
SBIS	P,b	Пропустити якщо біт у порту встановлений	if(I/O(P,b)==1) PC=PC + 2 or 3	None	1/2/3
BRBC	s,k	Перейти якщо прапор у SREG очищений	if(SREG(s)==0) PC=PC+ k + 1	None	1/2
BRBS	s,k	Перейти якщо прапор у SREG установлений	if(SREG(s)==1) PC = PC+k+ 1	None	1/2
BREQ	k	Перейти якщо дорівнює	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Перейти якщо не дорівнює	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Перейти якщо перенос установлений	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Перейти якщо перенос очищений	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Перейти якщо дорівнює чи більше	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Перейти якщо менше	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Перейти якщо мінус	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Перейти якщо плюс	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Перейти якщо більше чи дорівнює (зі знаком)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Перейти якщо менше (зі знаком)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Перейти якщо прапор внутрішнього переносу встановлений	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти якщо прапор внутрішнього переносу очищений	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти якщо прапор T встановлений	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти якщо прапор T очищений	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Перейти якщо прапор переповнення встановлений	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти якщо прапор переповнення очищений	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти якщо переривання дозволені	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти якщо переривання заборонені	if(I==0) PC = PC + k + 1	None	1/2

Виконувати арифметико-логічні операції й операції читання безпосередньо над змістом комірок пам'яті не можна. Не можна також записати константу чи очистити вміст комірки пам'яті.

Система команд AVR дозволяє лише виконувати операції обміну даними між осередками SRAM і регістрами загального призначення.

Перевагами системи команд можна вважати різноманітні режими адресації комірок пам'яті.

Усі регістри введення/виведення можуть зчитуватися і записуватися через регістри загального призначення за допомогою команд IN, OUT.

Безпосередня установка і скидання окремих розрядів цих регістрів виконується командами SBI і CBI. Команди умовних переходів у якості своїх операндів можуть мати як біти-ознаки результату операції, так і окремі розряди регістрів введення/виведення, що побітно адресуються.

Інструкції передачі даних

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
MOV	Rd,Rr	Скопіювати регістр	Rd = Rr	None	1
LDI	Rd,K8	Завантажити константу	Rd = K	None	1
LDS	Rd,k	Пряме завантаження	Rd = (k)	None	2*
LD	Rd,X	Непряме завантаження	Rd = (X)	None	2*
LD	Rd,X+	Непряме завантаження з пост-інкрементом	Rd=(X), X=X+1	None	2*
LD	Rd,-X	Непряме завантаження з пре-декрементом	X=X-1, Rd=(X)	None	2*
LD	Rd,Y	Непряме завантаження	Rd = (Y)	None	2*
LD	Rd,Y+	Непряме завантаження з пост-інкрементом	Rd=(Y),Y=Y+1	None	2*
LD	Rd,-Y	Непряме завантаження з пре-декрементом	Y=Y-1,Rd= (Y)	None	2*
LDD	Rd,Y+q	Непряме завантаження з заміщенням	Rd = (Y+q)	None	2*
LD	Rd,Z	Непряме завантаження	Rd = (Z)	None	2*
LD	Rd,Z+	Непряме завантаження з пост-інкрементом	Rd= (Z), Z=Z+1	None	2*
LD	Rd,-Z	Непряме завантаження з пре-декрементом	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Непряме завантаження з заміщенням	Rd = (Z+q)	None	2*
STS	k,Rr	Пряме збереження	(k) = Rr	None	2*
ST	X,Rr	Непряме збереження	(X) = Rr	None	2*
ST	X+,Rr	Непряме збереження з пост-інкрементом	(X)=Rr, X=X+1	None	2*
ST	-X,Rr	Непряме збереження з пре-декрементом	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Непряме збереження	(Y) = Rr	None	2*
ST	Y+,Rr	Непряме збереження з пост-інкрементом	(Y)=Rr, Y=Y+1	None	2
ST	-Y,Rr	Непряме збереження з пре-декрементом	Y=Y-1, (Y)= Rr	None	2
ST	Y+q,Rr	Непряме збереження з заміщенням	(Y+q) = Rr	None	2
ST	Z,Rr	Непряме збереження	(Z) = Rr	None	2
ST	Z+,Rr	Непряме збереження з пост-інкрементом	(Z)= Rr, Z=Z+1	None	2
ST	-Z,Rr	Непряме збереження з пре-декрементом	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Непряме збереження з заміщенням	(Z+q) = Rr	None	2
LPM	Нет	Завантаження з програмної пам'яті	R0 = (Z)	None	3
LPM	Rd,Z	Завантаження з програмної пам'яті	Rd = (Z)	None	3
LPM	Rd,Z+	Завантаження з програмної пам'яті з пост-інкрементом	Rd=(Z), Z=Z+1	None	3
SPM	Нет	Збереження в програмній пам'яті	(Z) = R1:R0	None	-
IN	Rd,P	Читання порту	Rd = P	None	1
OUT	P,Rr	Запис у порт	P = Rr	None	1
PUSH	Rr	Занесення регістра в стек	STACK = Rr	None	2
POP	Rd	Витяг регістра зі стека	Rd = STACK	None	2

Інструкції роботи з бітами

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
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	Логічне зрушення вправо	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Циклічне зрушення вліво через C	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Циклічне зрушення вправо через C	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Арифметичне зрушення вправо	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Перестановка тетрад	Rd(3..0)=Rd(7..4),Rd(7..4)=Rd(3..0)	None	1
BSET	s	Установка прапора	SREG(s) = 1	SREG(s)	1
BCLR	s	Очищення прапора	SREG(s) = 0	SREG(s)	1
SBI	P,b	Установити біт у порту	I/O(P,b) = 1	None	2
CBI	P,b	Очистити біт у порту	I/O(P,b) = 0	None	2
BST	Rr,b	Зберегти біт з регістра в T	T = Rr(b)	T	1
BLD	Rd,b	Завантажити біт з T у регістр	Rd(b) = T	None	1
SEC	Hi	Установити прапор переносу	C =1	C	1
CLC	Hi	Очистити прапор переносу	C = 0	C	1
SEN	Hi	Установити прапор негативного числа	N = 1	N	1
CLN	Hi	Очистити прапор негативного числа	N = 0	N	1
SEZ	Hi	Встановити прапор нуля	Z = 1	Z	1
CLZ	Hi	Очистити прапор нуля	Z = 0	Z	1
SEI	Hi	Встановити прапор переривань	I = 1	I	1
CLI	Hi	Очистити прапор переривань	I = 0	I	1
SES	Hi	Установити прапор числа зі знаком	S = 1	S	1
CLN	Hi	Очистити прапор числа зі знаком	S = 0	S	1
SEV	Hi	Установити прапор переповнення	V = 1	V	1
CLV	Hi	Очистити прапор переповнення	V = 0	V	1
SET	Hi	Установити прапор T	T = 1	T	1
CLT	Hi	Очистити прапор T	T = 0	T	1
SEH	Hi	Установити прапор внутрішнього переносу	H = 1	H	1
CLH	Hi	Очистити прапор внутрішнього переносу	H = 0	H	1
NOP	Hi	Немає операції	Hi	None	1
SLEEP	Hi	Спати (зменшити енергоспоживання)	Дивитися опис інструкції	None	1
WDR	Hi	Скидання сторожового таймера	Дивитися опис інструкції	None	1

Асемблер не розрізняє регістр символів. Операнди можуть бути таких видів:

- Rd: результуючий і вихідний регістр;
- Rr: вихідний регістр;

- b: константа (3 біти), може бути константний вираз;
- s: константа (3 біти), може бути константний вираз;
- P: константа (5-6 біт), може бути константний вираз;
- K6: константа (6 біт), може бути константний вираз;
- K8: константа (8 біт), може бути константний вираз;
- k: константа, може бути константний вираз;
- q: константа (6 біт), може бути константний вираз;
- Rdl: R24, R26, R28, R30 для інструкцій ADIW і SBIW;
- X,Y,Z: регістри непрямої адресації (X=R27:R26, Y=R29:R28, Z=R31:R30).