
Albert Kotvytskiy
Intellectual capital
is the foundation of
innovative development
'2024
Robotic systems
«European Science»
Book 28. Part 1

SWorld
Germany

Albert Kotvytskiy
Котвицький А. Т.

INTELLEKTUELLES KAPITAL - DIE GRUNDLAGE FÜR INNOVATIVE ENTWICKLUNG

ROBOTISCHE SYSTEME

INTELLECTUAL CAPITAL IS THE FOUNDATION OF INNOVATIVE DEVELOPMENT

ROBOTIC SYSTEMS

Monographic series «European Science»
Book 28. Part 1.

In internationalen wissenschaftlich-geometrischen Datenbanken enthalten
Included in International scientometric databases

MONOGRAPHIE

MONOGRAPH

ScientificWorld-NetAkhatAV
Karlsruhe 2024

Authors / Автори:
Albert Kotvytskiy / *Котвицький А. Т.*

Intellektuelles Kapital - die Grundlage für innovative Entwicklung: Robotische Systeme. Monografische Reihe «Europäische Wissenschaft». Buch 28. Teil 1. 2024.

Intellectual capital is the foundation of innovative development: Robotic systems. Monographic series «European Science». Book 28. Part 1. 2024.

ISBN 978-3-98924-041-4
DOI: 10.30890/2709-2313.2024-28-01

Published by:
ScientificWorld-NetAkhatAV
Lußstr. 13
76227 Karlsruhe, Germany
e-mail: *editor@promonograph.org*
site: https://desymp.promonograph.org

Inhalt / Content

Зміст

Вступ

У цьому посібнику розглянуто матеріал необхідний розуміння принципів управління робототехнічними системами за допомогою мікроконтролерів. При проектуванні вбудованих систем основними завданнями є отримання даних із датчиків, аналіз отриманих сигналів, генерування керуючих сигналів для периферійних пристроїв. Найпростішим прикладом датчика може бути звичайна кнопка. Робототехнічна система повинна вловити натискання кнопки та на підставі цього виконати ті чи інші дії. По суті, датчиком є будь-який пристрій, який отримує сигнал із реального фізичного світу. Прикладами датчиків можуть бути як найпростіші аналогові пристрої: кнопки, термометри, фоторезистори. Так і складніші цифрові пристрої такі як: акселерометри, інфрачервоні датчики руху, відеокамери. Тому первинним завданням робототехніки є реєстрація цифрового або аналогового сигналу. Потім настає стадія обробки отриманого сигналу та прийняття рішення на основі цієї інформації. Алгоритми прийняття рішень - це окрема велика область робототехніки, яка починається від найпростіших операторів розгалуження до алгоритмів комп'ютерного зору, планування руху та машинного навчання. Після того, як рішення прийнято, його потрібно реалізувати. Це досягається за допомогою актуаторів. Актуатор це виконавчий пристрій або його активний елемент, що перетворює один вид енергії на інший, що призводить до виконання певної дії. Найпростішим актуатором можна вважати звичайний світлодіод, який перетворює електричну енергію на світлову і може служити для різних цілей, таких як індикації поточного стану, сигналізації помилок, зміні рівня освітленості робочого простору і так далі. Стандартно до актуаторів відносять такі пристрої, як двигуни постійного та змінного струмів, сервоприводи, крокові двигуни, електромагнітні клапани, лінійні приводи і так інше. Управління актуаторами дозволяє роботу переміщатися; піднімати, опускати, захоплювати предмети; обертатися та виконувати безліч інших дій.

Даний посібник дозволить студентам отримати знання та навчитися їх практично використовувати при створенні робототехнічних систем. Представлений курс розроблено таким чином, що дозволяє це зробити як на реальному обладнанні, так і в програмах симуляції.

Основними робочими інструментами в цьому курсі будуть два мікроконтролери: ATmega328 і ATmega2560. На першому з них побудовано відому платформу Arduino UNO, а на другому – Arduino MEGA. Для аналізу генерованих сигналів ми будемо використовувати логічний аналізатор або його програмну симуляцію.

По суті, одним із найважливіших завдань у робототехніці є завдання генерування необхідних та аналізу отриманих сигналів. Для цього, в цій частині навчального курсу, ми вивчатимемо наступні апаратні засоби мікроконтролерів: порти вводу/виводу, 8-ми та 16-бітові таймер-лічильники, різні способи управління за допомогою переривань, протокол USART (UART) та модуль АЦП.

Автор висловлює величезну подяку за технічну та інформаційну підтримку словацькому університету імені П.Й. Шафарика в Кошицях, грант EU NextGenerationEU через Recovery and Resilience Plan for Slovakia under project No. 09I03-03-V01-00119.

Лабораторна робота №1
Обладнання та програмне забезпечення

Мета: познайомитися з необхідним обладнанням та програмним забезпеченням.

Для вивчення основ робототехніки нам знадобиться як реальне обладнання з необхідним програмним забезпеченням, так і програми для симуляції електронних схем і мікроконтролерів.

Спочатку розглянемо реальне устаткування.

В цьому курсі ми будемо вивчати базові концепції керування робототехнічними системами на прикладі мікроконтролерів ATmega328 та ATmega2560. Для того щоб зосередитись безпосередньо на можливостях цих мікроконтролерів та для простоти підключення і програмування будемо використовувати платформи Arduino UNO (ATmega328 див. рисунок 1.1) та Arduino MEGA (ATmega2560 див. рисунок 1.2).

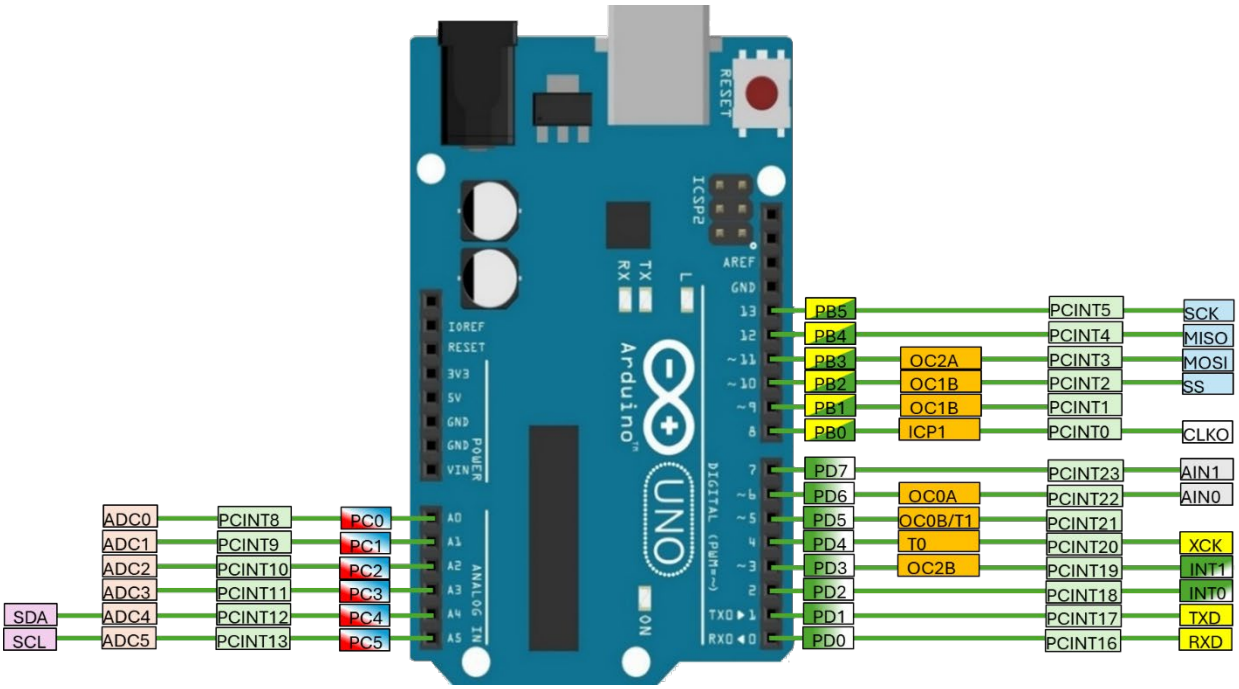


Рисунок 1.1 - Мікроконтролер ATmega328 на платформі Arduino UNO.

Для програмування вказаних Arduino платформ нам знадобиться програмне середовище Arduino IDE яке можна вільно завантажити та встановити на своєму комп'ютері. Відповідність між ніжками мікроконтролерів та пінів на платформі Arduino представлено на рис. 1 та рис.2. Пояснимо один дуже важливий момент.

Цифрові піни платформи Arduino будемо позначати як D0, D1 і так далі, аналогові піни – A0, A1 …

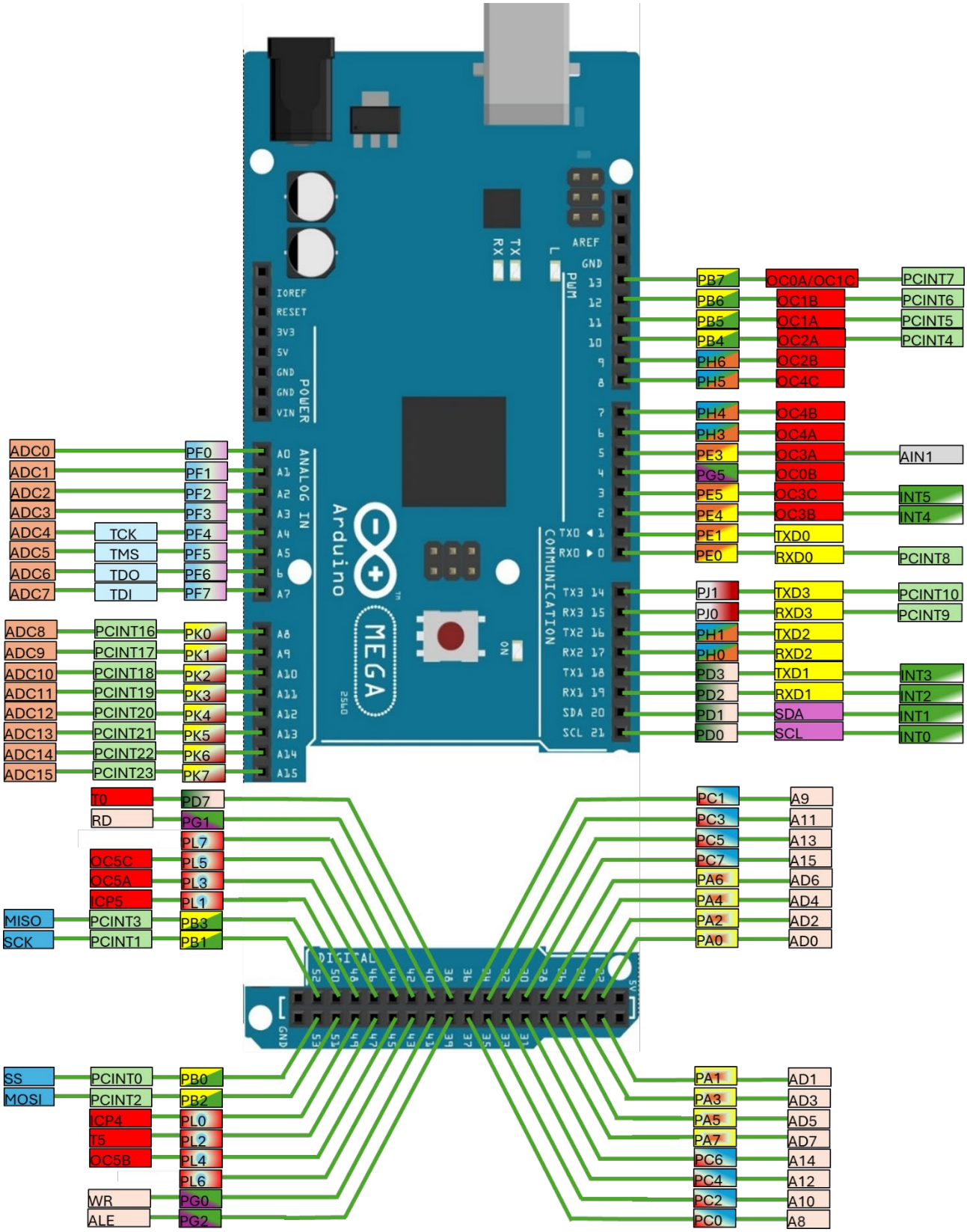


Рисунок 1.2 - Мікроконтролер ATmega2560 на платформі Arduino MEGA.

Таким чином, наприклад, пін D8 на платформі Arduino UNO відповідає піну PB0 мікроконтролера ATmega328, а той же самий пін D8 платформи Arduino MEGA відповідає піну PH5 мікроконтролера ATmega2560. Також пін A0 на Arduino UNO відповідає піну PC0 мікроконтролера ATmega328, а на Arduino MEGA – PF0 (дивись рисунки 1 та 2).

Платформи Arduino з'єднуються з комп'ютером за допомогою кабелю типу USB A-B по якому відбувається живлення мікроконтролерів та необхідної периферії, а також програмування.

Макетна плата

Якщо згідно з завданням лабораторної роботи треба зібрати електричну схему, це потрібно робити на макетній платі. В лабораторії рекомендовано використовувати плату на 830 точок з кроком 2,54 мм (0,1 дюйма).

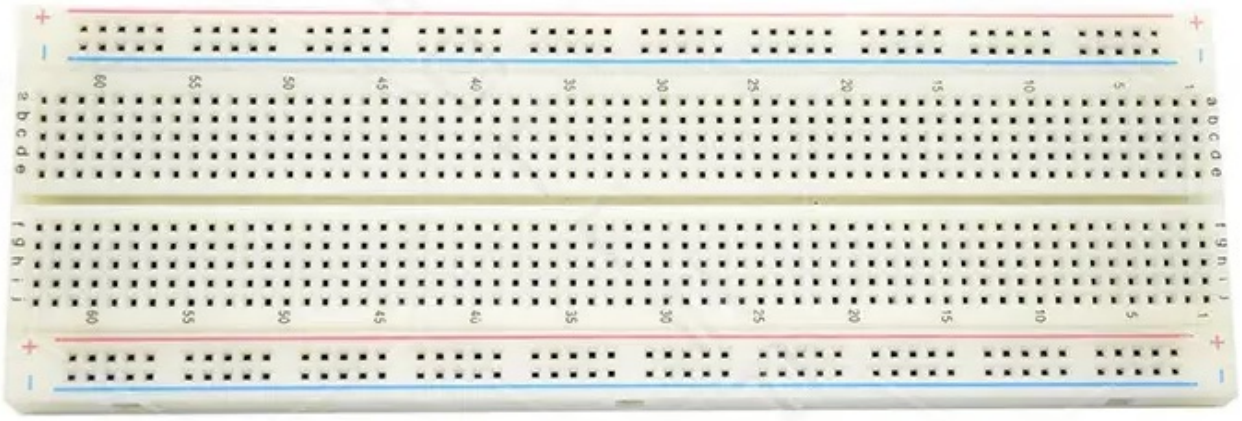


Рисунок 1.3 - Зовнішний вигляд макетної плати.

В отворах розташовані контактні роз'єми для встановлення деталей з контактами діаметром до 0,8 мм. Макетна плата має дві шини живлення (горизонтальні рядки які позначені синьою «-» і червоною «+» лініями) та вертикальні рядки з'єднані по 5 штук. Роз'єми та шини виконані у вигляді металевих контактів, вставлених зі зворотного боку плати, та закритих захисною наклейкою. На рисунку 1.4 зображено з'єднання роз'ємів.

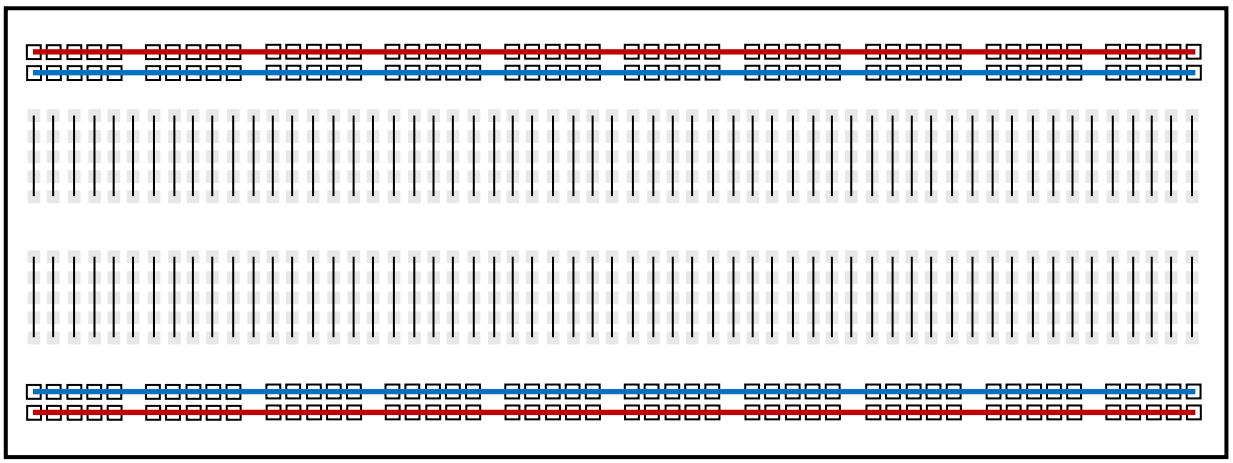


Рисунок 1.4 - Структура макетної плати.

Резистори та їх кольорове маркування

Для визначення номіналу резисторів слід скористатися наступною таблицею.

Таблиця 1.1 – Кольорова схема для визначення опору резистора

Color	1st band	2nd band	3rd band	Multiplier	Tolerance
Black	0	0	0	×1	
Brown	1	1	1	×10	±1%
Red	2	2	2	×100	±2%
Orange	3	3	3	×1000	
Yellow	4	4	4	×10 000	
Green	5	5	5	×100 000	±0.5%
Blue	6	6	6	×1000 000	±0.25%
Violet	7	7	7	×10 000 000	±0.10%
Grey	8	8	8		±0.05%
White	9	9	9		
Gold				×0.1	±5%
Silver				×0.01	±10%

За допомогою цієї таблиці можна визначати номінал резисторів які маркіруються кольоровими кільцями. Але ми здебільше будемо користуватись резисторами на 510 Ом та на 10 кОм (дивись рисунок 1.5) на якому представлено кольорове

маркування резисторів, які використовуються нами в лабораторних роботах, і приклад розрахунку їхнього номіналу.

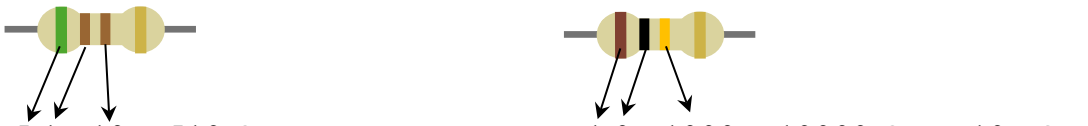


Рисунок 1.5 - Резистори на 510 Ом та 10 кОм.

Тому рекомендується запам'ятати як виглядають ці резистори.

Конденсатори

Також в цьому курсі ми будемо використовувати керамічні конденсатори на 0,1 мкФ який маркірується як 104, та на 1 мкФ – маркіровка 105 (дивись рисунок 1.6).

Рисунок 1.6 - Конденсатори.

Код 104 означає 10×10000 пФ = 10^5 пФ = 0,1 мкФ, таким же чином розшифровується код 105, маємо 10×100000 пФ = 10^6 пФ = 1 мкФ.

Кнопка

На рисунку 1.7 ми бачимо зовнішній вигляд кнопки та її принципову електричну схему. Звідси випливає, що ми можемо використовувати будь яку пару не замкнених виводів.

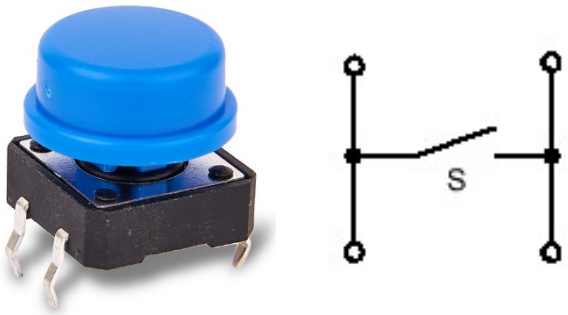


Рисунок 1.7 - Кнопка та її принципова електрична схема.

Світлодіод

Для наочності керування GPIO мікроконтролера дуже часто використовують світлодіоди. Нагадаємо, що світлодіоди завжди підключаються до пінів мікроконтролера через обмежуючий струм резистор, для цього ми будемо використовувати резистор на 510 Ом. Для того щоб світлодіод випромінював світло анод світлодіоду потрібно підключити до високого, а катод до низького потенціалів. Анод світлодіоду має більшу довжину ніж катод (див. рисунок 1.8), або колба світлодіоду зі сторони катоду має зріз. Позначення світлодіоду на принциповій електричній схемі також зображено на рисунку 1.8.

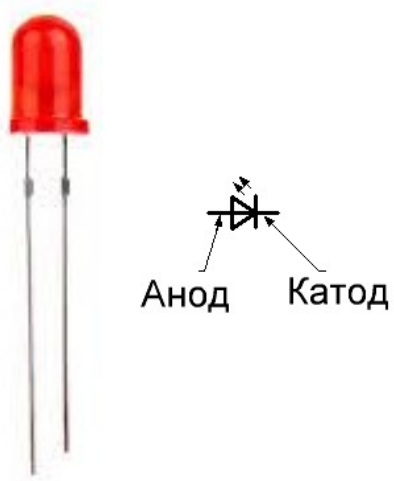


Рисунок 1.8 - Світлодіод та його принципова схема.

Потенціометр

При вивченні аналого-цифрового перетворювача нам знадобиться потенціометр або змінний резистор. У нашому курсі рекомендується використовувати потенціометр від 5 до 20 кОм. Наприклад, такий який зображений рисунку 1.9, такий потенціометр легко вставляється в макетну плату та дозволяє проводити його підключення без паяння.

Рисунок 1.9 -Змінний резистор на 5 кОм

Логічний аналізатор

Для розуміння того, що відбувається на пінах мікроконтролерів зручно використовувати логічний аналізатор. В наших лабораторних роботах ми будемо користуватись восьми-канальним USB логічним аналізатором Logic Saleae (див. рисунок 1.10) із максимальною частотою дискретизації 24 мегасемплов у секунду. Для його використання спочатку потрібно встановити програмне забезпечення (saleae.com). Потім під'єднати аналізатор до комп'ютера за допомогою кабелю USB – mini USB. Якщо програмне забезпечення встановилось вірно, то відбудеться автоматичне розпізнавання підключеного обладнання (див. рисунок

Рисунок 1.10 - Логічний 8-ми канальний аналізатор Logic Saleae

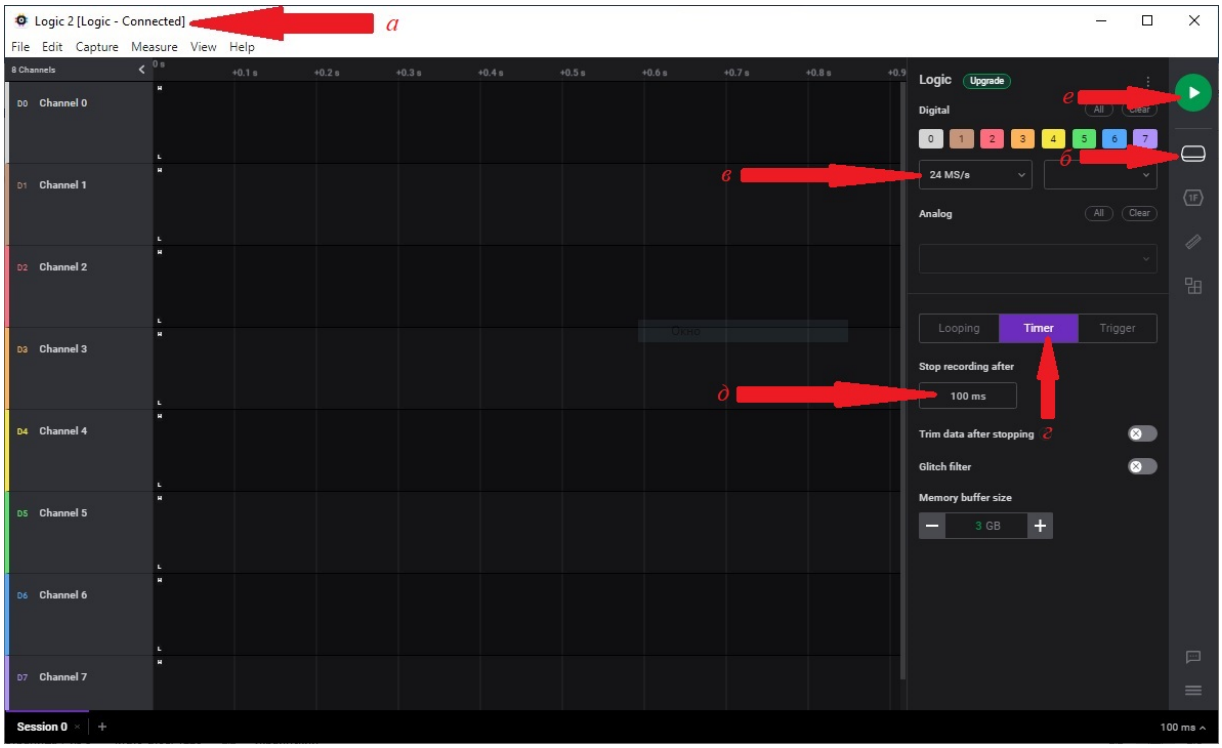


Рисунок 1.11 - Рекомендовані налаштування логічного аналізатору.

1.11, стрілка *а*). Для встановлення параметрів роботи програми натисніть кнопку яка позначена на рисунку 1.11 червоною стрілкою з літерою *б*. За замовчуванням частота роботи аналізатора дорівнює 24 MS/s (див. рисунок 1.11, стрілка *в*). Далі виберіть вкладку Timer (див. рисунок 1.11, стрілка *г*) та в полі Stop recording after

замість 100 секунд введіть 0.1 та натисніть enter. Після цього в даному полі автоматично встановиться значення 100 мілісекунд (див. рисунок 1.11, стрілка *д*). Для того, щоб почати захоплення даних потрібно натиснуту зелену кнопку (див. рисунок 1.11, стрілка *е*). Але спочатку необхідно відключити аналізатор від комп'ютера, з'єднати пін GND аналізатора з піном GND мікроконтролера, а також під'єднати необхідні піни каналів CH1 – CH8 аналізатора до потрібних пінів мікроконтролера та завантажити в нього програму.

Робота в Proteus

Після встановлення програми Proteus 8.x та її запуску бачимо наступне вікно (рисунок 1.12).

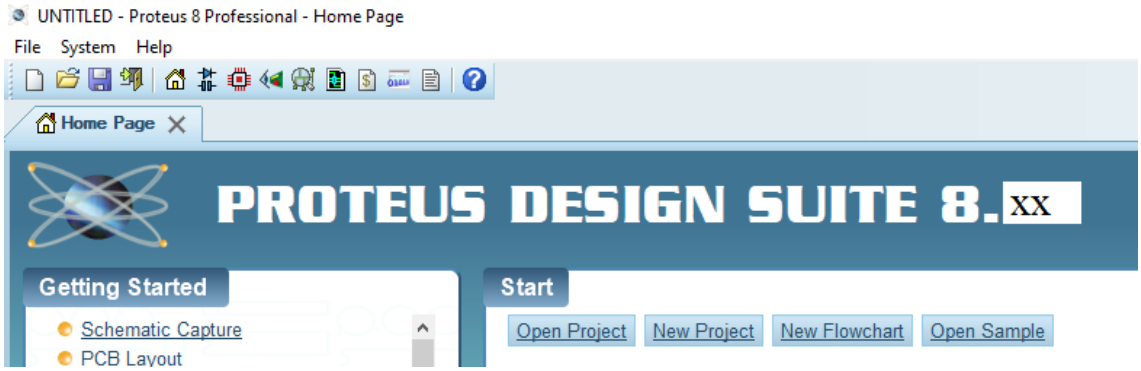


Рисунок 1.12 - Вікно запрошення Proteus

Обираємо New Project та отримуємо наступне вікно (рисунок 1.13).

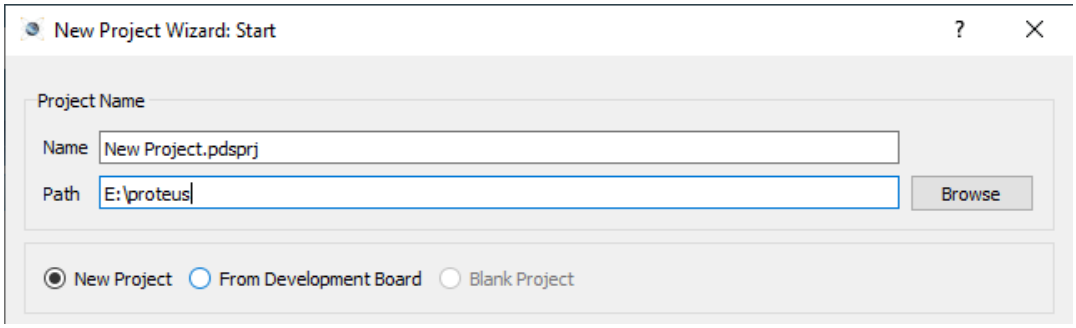


Рисунок 1.13 - Створення нового проекту

У полі Name вводимо ім'я свого проекту, у полі Path вибираємо папку, в якій буде збережено проект.

Далі є дві можливості: New Project і From Development Board.

Спочатку розглянемо найпростіший старт роботи. Для цього обираємо From Development Board та отримуємо наступне вікно (рисунок 1.14):

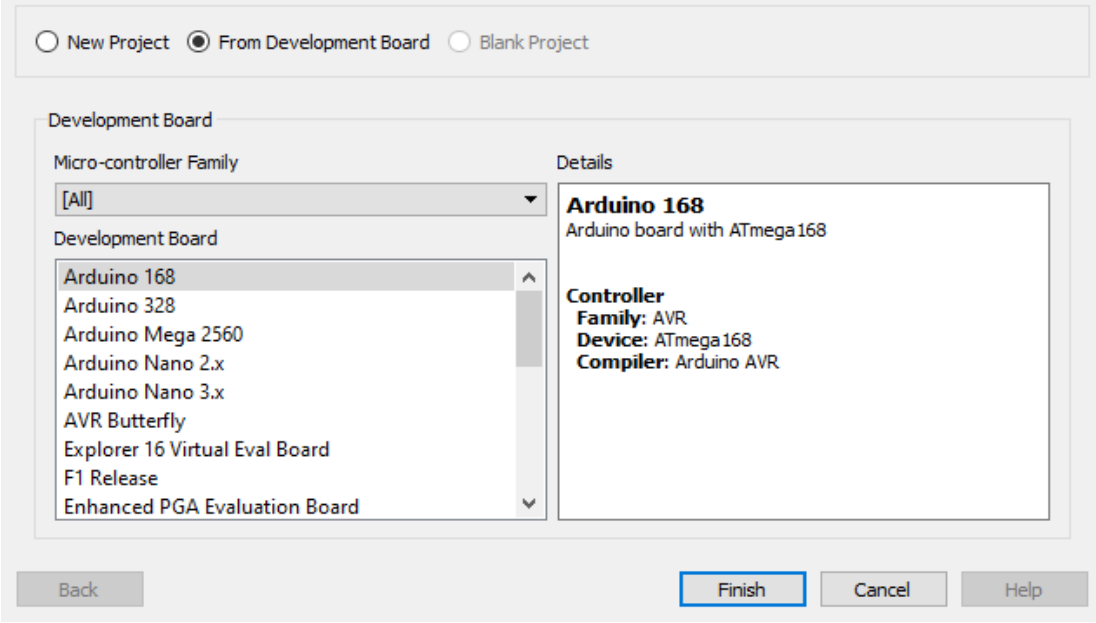


Рисунок 1.14 - Вибір From Development Board

У нашому курсі розглядається програмування двох мікроконтролерів ATmega328 та ATmega2560. Тому вибираємо той, який заданий викладачем, наприклад, Arduino Mega2560, натискаємо Finish і отримуємо наступний робочий простір (рисунок 1.15):

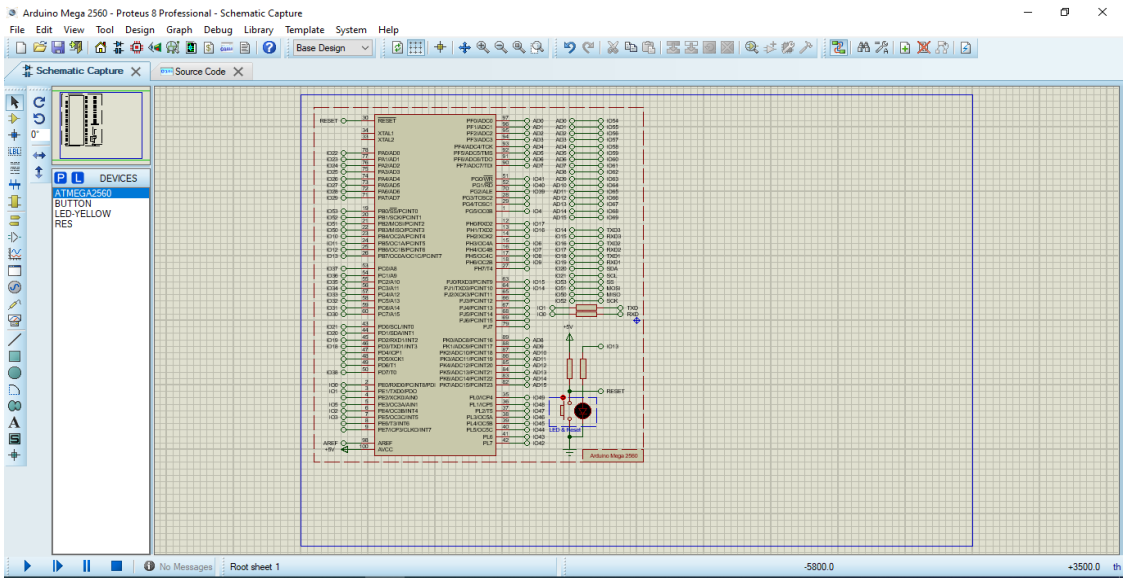


Рисунок 1.15 - Початковий робочий простір з Arduino MEGA

Тут бачимо дві вкладки 

. Перша вкладка Schematic Capture, призначена для створення електричної схеми, друга – Source Code – для написання програми. Для початку переконаємося, що все встановилося коректно

та Proteus працездатний. Для цього перейдемо на вкладку Source Code і побачимо, що Proteus налаштував початковий код у стилі Arduino IDE на мові WIRING

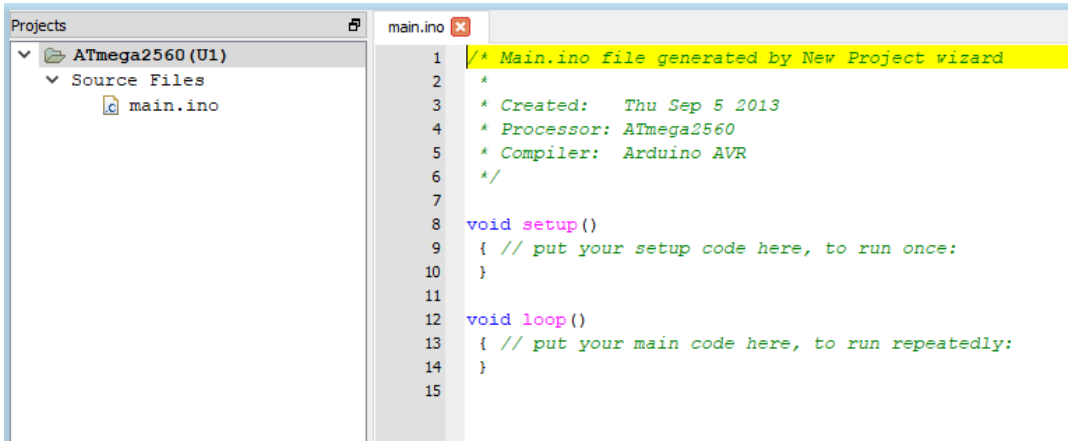


Рисунок 1.16 - Шаблон програми у стилі Arduino IDE

Видалимо всі коментарі та напишемо стандартну програму миготіння вбудованим світлодіодом який підключено до піна D13. Пояснимо один момент. Ми будемо писати програми на «чистому» CI, але іноді для тестування можемо використовувати мову WIRING.

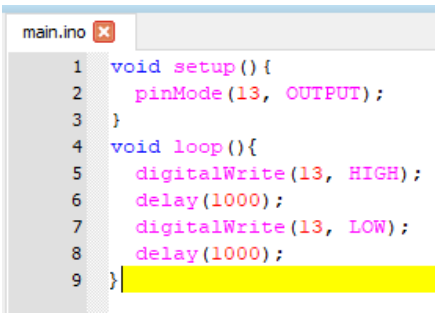


Рисунок 1.17 - Базова програма миготіння світлодіодом

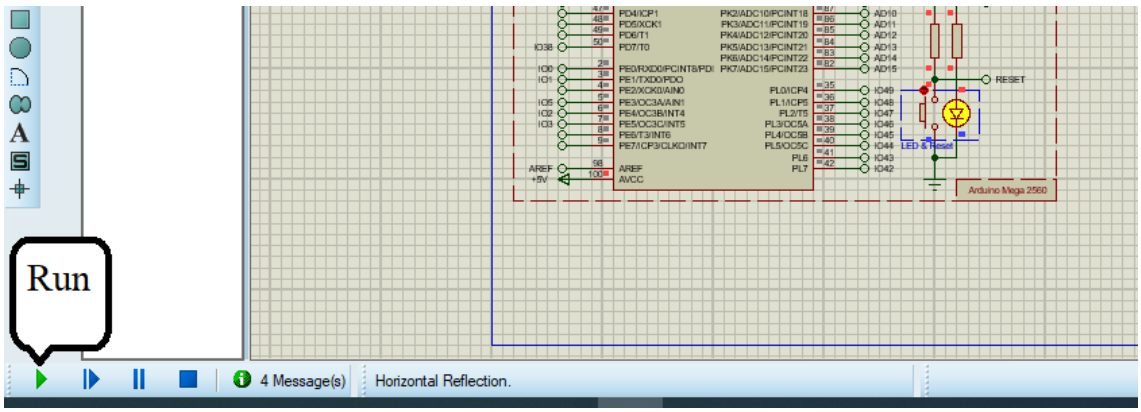


Рисунок 1.18 - Запуск симуляції в Proteus

Після натискання кнопки Run (рисунок 1.18) починається симуляція: світлодіод починає блимати жовтим кольором.

Вбудоване в Proteus середовище розробки Arduino IDE також, як і реальне середовище розробки, дозволяє писати програму на CI. Для того, щоб у цьому

переконатися, напишемо програму миготіння світлодіодом на піні D13, але вже без використання «ардуіновських функцій». Поки, не вдаючись у подробиці, завантажимо код **Prog1.2**.

Prog1.2

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void) {
 DDRB  |= (1<<DDB7);
 while(1) {
    PORTB ^= (1<<PORTB7);
    _delay_ms(1000);
 }
}
```

Отримуємо наступний вигляд (рисунок 1.19).

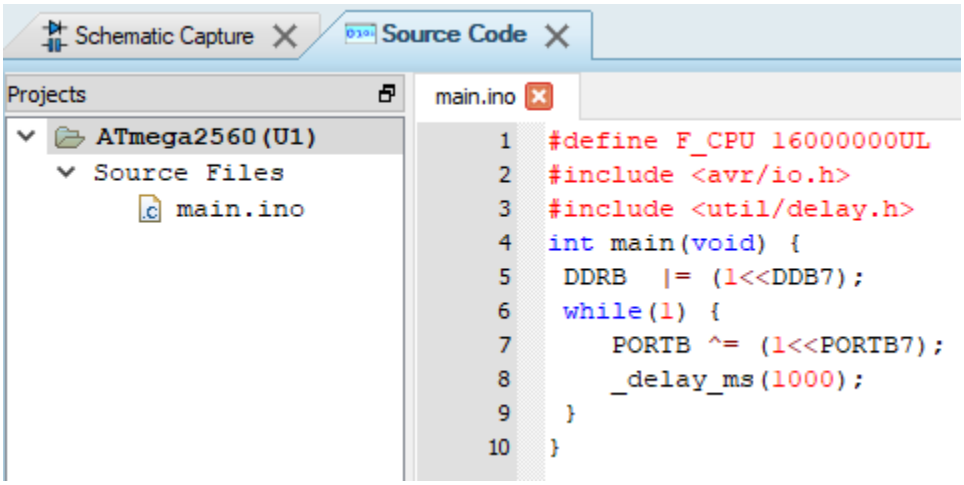


Рисунок 1.19 - Програма на CI в Proteus

Після запуску симуляції (синій трикутник знизу зліва) побачимо, що підключений світлодіод одну секунду світиться і одну не світиться.

У нашому курсі нам доведеться часто аналізувати отримані сигнали. Для цього ми користуватимемося віртуальним приладом, а саме – логічним аналізатором. Додаймо його у робоче поле.

Для цього спочатку натиснемо на кнопку вибору віртуального вимірювального приладу, а потім з списку обираємо логічний аналізатор (рисунок 1.20). Після чого переведемо покажчик миші на робоче поле та встановимо на ньому логічний аналізатор.

Розглядаючи Arduino MEGA 2560 у Proteus, знаходимо, що вивід PB7 (саме ним ми керуємо у програмі) відповідає IO13 (input-ouput 13) або D13 в ідеології

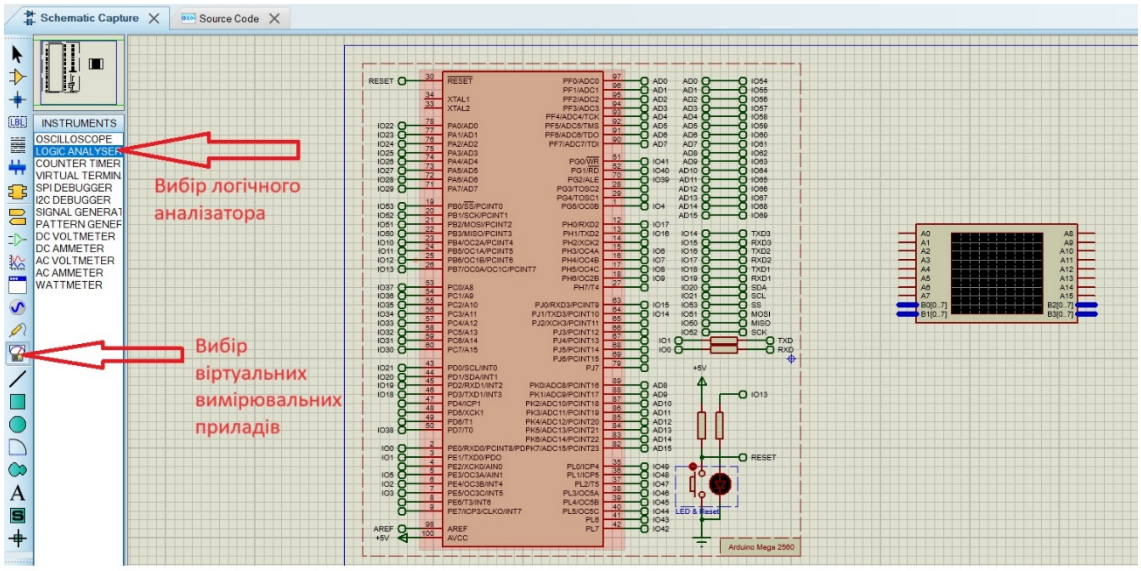
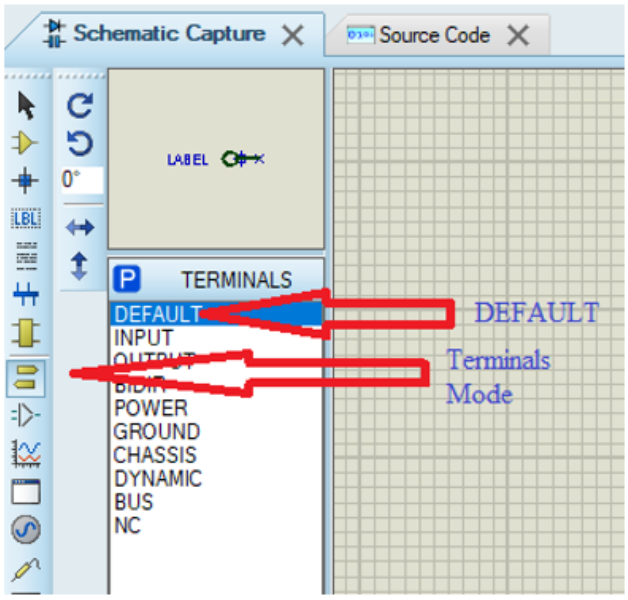


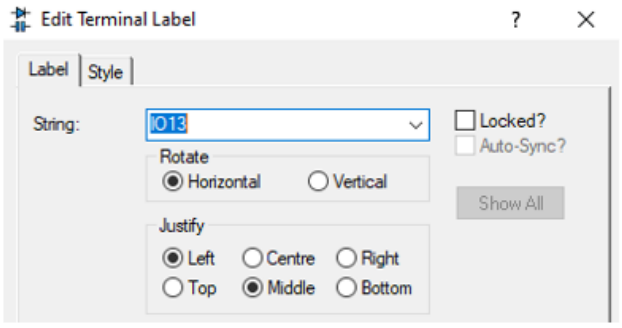
Рисунок 1.20 - Додавання логічного аналізатора

Arduino. Підключимо вивід логічного аналізатора A0 до IO13 у такий спосіб. Натискаємо в лівій колонці кнопку Terminals Mode, а після цього зі списку, що з'явився, вибираємо DEFAULT (рисунок 1.21). Після цього маємо вказівник миші на робочому полі в потрібному місці (недалеко від логічного аналізатора) і натискаємо ліву кнопку. Повинно вийти приблизно так як показано на рисунку 1.22.



а)

Рисунок 1.22 - Встановлення конектора



б)

Рисунок 1.21 - Використання конектору для з'єднання пінів

Після цього натискаємо ще раз лівою кнопкою миші по встановленому об'єкту і пишемо в полі String назву піна IO13 (або вибираємо цю назву зі списку) та натискаємо кнопку OK. Тепер підводимо покажчик миші до клеми та з'єднуємо її з виведенням A0 логічного аналізатора. Тепер схема в Proteus виглядає так (рисунок 1.23).

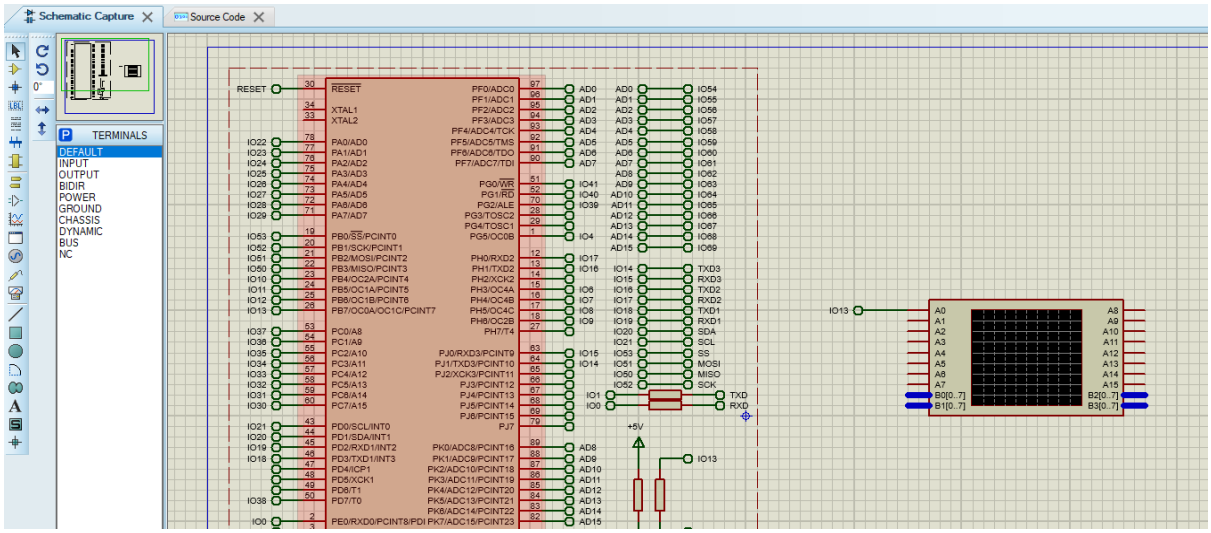


Рисунок 1.23 - Додавання логічного аналізатора

Інтервал часу в одну секунду, який ми задали в **Prog1.2**, досить великий для логічного аналізатора (хоча він може вимірювати час до 4 секунд). Змінимо зараз **Prog1.2** на **Prog1.3.**

Prog1.3

```
#define F_CPU 16000000UL     // Частота - 16MHz
#include <avr/io.h>          // Бібліотека вводу/виводу
#include <util/delay.h>      // Бібліотека затримок
int main(void) {
 DDRB  |= (1<<DDB7);       // Встановлюємо пін PB7=D13 mega2560 на ВИХІД
 while(1) {
    PORTB |= (1<<PORTB7); // Встановлюємо лог.1 на PB7
    _delay_ms(10);        // Затримка 10 мілісекунд
    PORTB &= ~(1<<PORTB7); // Встановлюємо лог.0 на PB7
    _delay_ms(5);          // Затримка 5 миллисекунд
  }
}
```

В **Prog1.3** встановлюємо логічну одиницю на піні PB7 на 10 мілісекунд, а логічний нуль на 5 мілісекунд. Тепер вкладка із програмою (Source Code) виглядає як

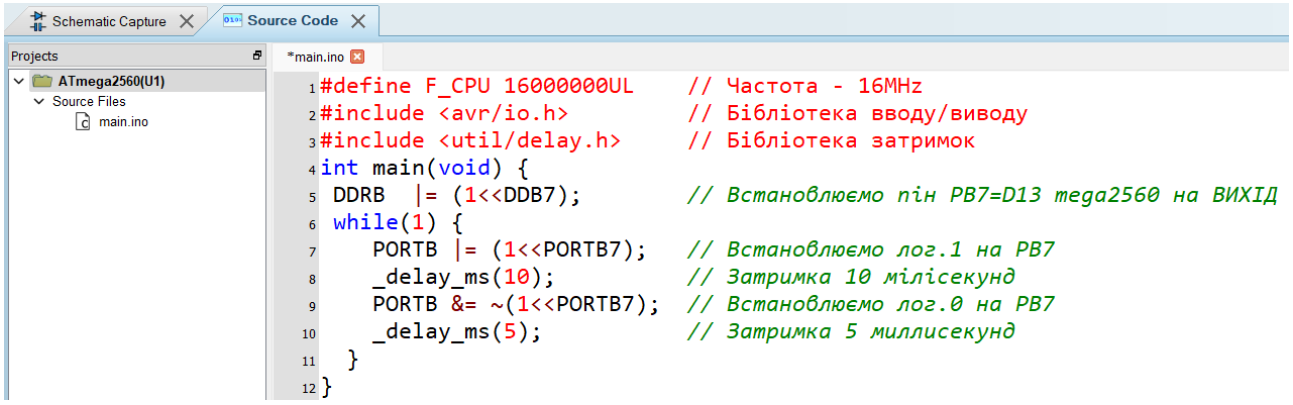


Рисунок 1.24 - Програма блимання світлодіодом на CI

Після запуску симуляції бачимо розгорнуте вікно логічного аналізатора (рисунок 1.25). *Не закривайте це вікно, воно закривається самостійно після зупинки симуляції. Якщо ви все-таки його закрили, пройдіть в пункт меню* Debug *і виберіть* VSM Logic Analyser.

Розберемо основні налаштування для роботи із цим інструментом.

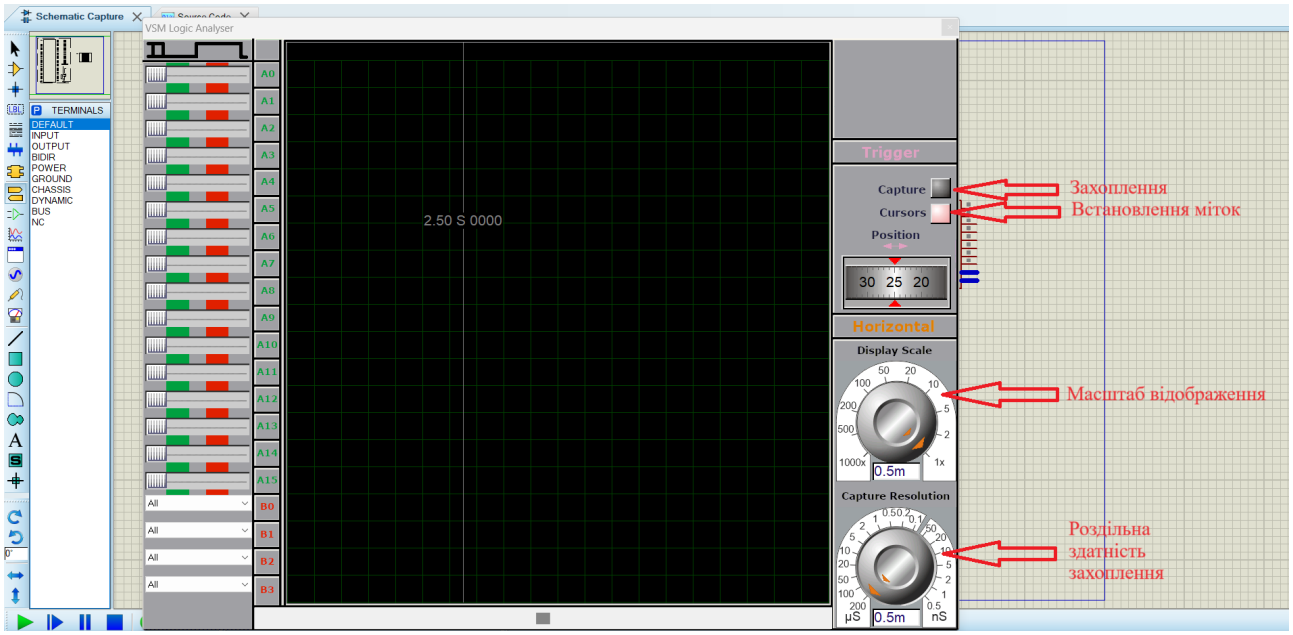


Рисунок 1.25 - Налаштування логічного аналізатора

У нижньому правому куті вікна логічного аналізатора ми бачимо налаштування: Display Scale і Capture Resolution. Почнемо з Capture Resolution - це роздільна здатність захоплення. Іншими словами, це час між яким відбувається зчитування та захоплення сигналу. Спочатку встановіть маленьку ручку (ручка тонкої настройки) цього регулятора в крайнє праве положення, тобто, поверніть її до упору за годинниковою стрілкою, а після цього, обертаючи

велику ручку, отримайте послідовно у відповідному полі значення: 0.5 ns = 0.5 наносекунд = 0.5×10^{-9} с

1 ns = 1 наносекунда = 1×10^{-9} с

2 ns = 2 наносекундb = 2×10^{-9} с

5 ns = 5 наносекунд = 5×10^{-9} с

10 ns = 10 наносекунд = 10×10^{-9} с

20 ns = 20 наносекунд = 20×10^{-9} с

50 ns = 50 наносекунд = 50×10^{-9} с

100 ns = 100 наносекунд = 100×10^{-9} с

0.2 us = 0.2 мікросекунд = 0.2×10^{-6} с

0.5 us = 0.5 мікросекунд = 0.5×10^{-6} с

1 us = 1 мікросекунда = 1×10^{-6} с

2 us = 2 мікросекунди = 2×10^{-6} с

5 us = 5 мікросекунд = 5×10^{-6} с

10 us = 10 мікросекунд = 10×10^{-6} с

20 us = 20 мікросекунд = 20×10^{-6} с

50 us = 50 мікросекунд = 50×10^{-6} с

0.1m = 0.1 мілісекунд = 0.1×10^{-3} с

0.2m = 0.2 мілісекунд = 0.2×10^{-3} с

Після того, як у вас все вийшло, встановіть будь-яке з цих значень, наприклад, 10 мікросекунд і покрутіть ручку тонкої настройки. У відповідному полі має змінюватися значення з 10 до 25 мікросекунд з досить маленьким кроком. Іншими словами, ручка тонкого налаштування задає множник від 1 до 2,5.

Тепер проведімо розрахункову оцінку сигналу, що надходить з мікроконтролера. Подивившись на **Prog1.3**, ми бачимо, що мінімальна тривалість сигналу, який нас цікавить, становить 5 мілісекунд. Для того, щоб надійно виміряти тривалість такого сигналу, потрібно, щоб час зчитування сигналу був щонайменше в 10 разів менше. Тобто максимальна тривалість у полі Capture Resolution має становити 0,5 мілісекунд, а краще набагато менше. Але

при цьому не варто забувати, що чим менше буде виставлено значення, тим більше буде навантажено систему. Для наших цілей цілком достатньо значення 0,1 мілісекунди.

Налаштування Display Scale встановлює масштаб відображення, тобто, налаштовує який часовий інтервал буде відповідати одній клітині екрана логічного аналізатора. Ширина цього екрану становить 20 клітинок. Тому, якщо тривалість низького рівня сигналу становитиме 4 клітини, то тривалість високого – 8 клітин. Тоді повний період вміститься у 12 клітин та ще 8 клітин залишиться. Таким чином, якщо 4 клітини відповідають 5 мілісекундам, то одна клітина буде відповідати тривалості 5/4=1.25 мілісекунд.

Після встановлення вибраних значень (дискову шкалу Position рекомендується встановлювати спочатку на 50%), потрібно натиснути кнопку захоплення Capture і почекати деякий час поки заповниться буфер, після цього на екрані з'явиться картина подібна до рисунку 1.27. Як ми й очікували, на каналі A0 з'явився сигнал, низький рівень якого відповідає 4 клітинам, а високий – 8. Швидше за все ви на своєму екрані побачите сигнал іншого кольору. Для того, щоб вибрати бажаний колір сигналу, потрібно на екрані логічного аналізатора натиснути праву кнопку миші і в меню (рисунок 1.26) обрати Colours Setup… Після цього побачимо вікно зображене на рисунку 1.28. В якому встановлюєте бажаний колір для виведення на монітор Display і для виведення на друк Print для потрібного каналу.

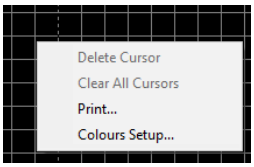


Рисунок 1.26 - Контекстне меню

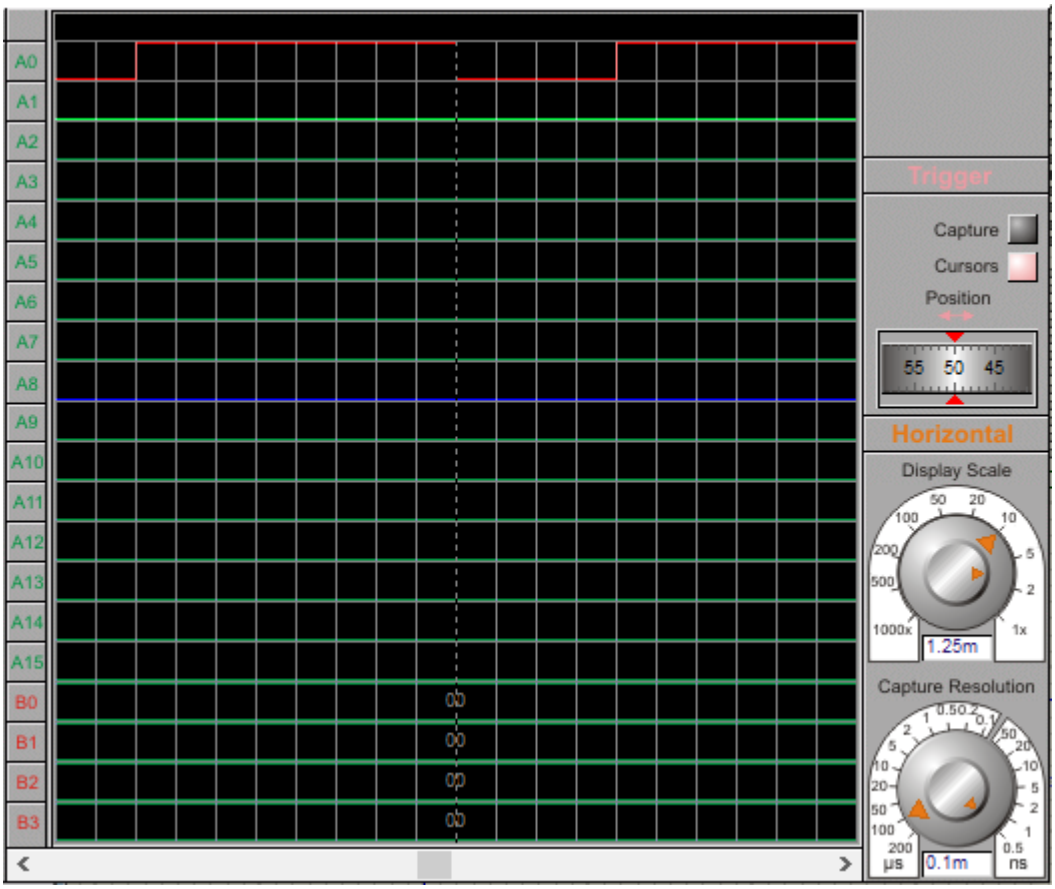


Рисунок 1.27 - Захоплення сигналу логічним аналізатором

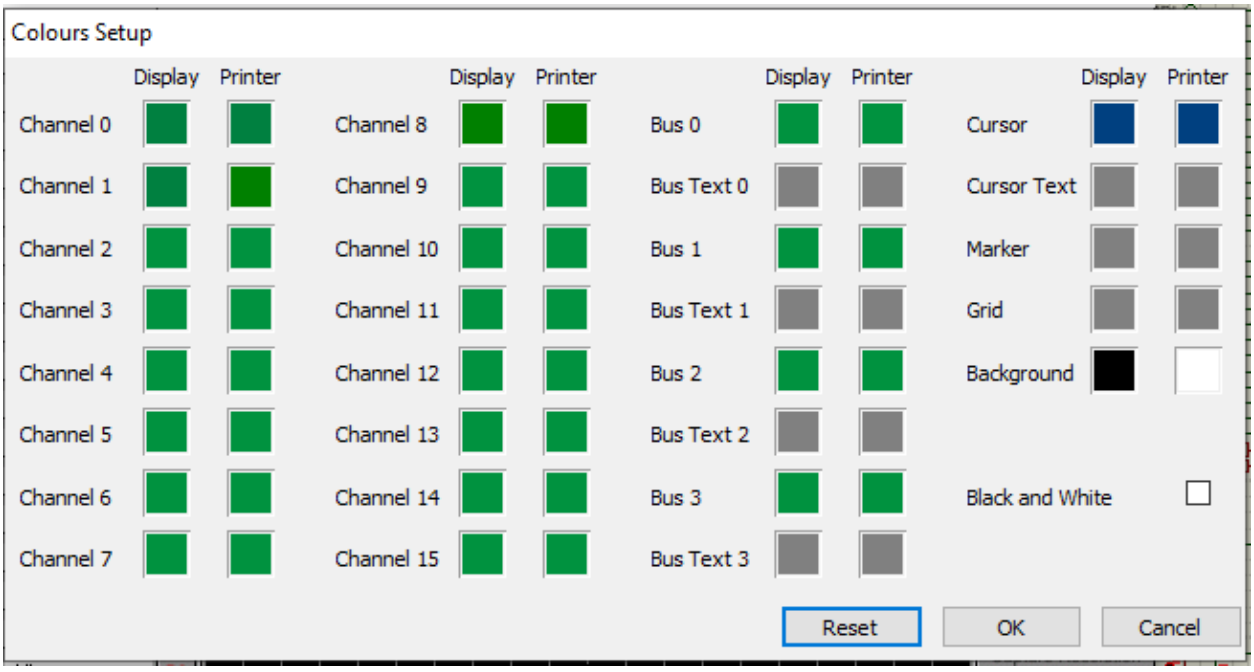


Рисунок 1.28 - Налаштування кольору для сигналу

Для більш точного вимірювання тривалості сигналів під час виконання лабораторних робіт необхідно встановити курсори чи мітки у потрібних місцях. Для цього натискаємо кнопку Cursors і після цього лівою кнопкою миші клацаємо необхідну кількість разів у потрібних місцях на екрані логічного аналізатора. Для аналізу **Prog1.3** можна встановити три курсори (рисунок 1.29).

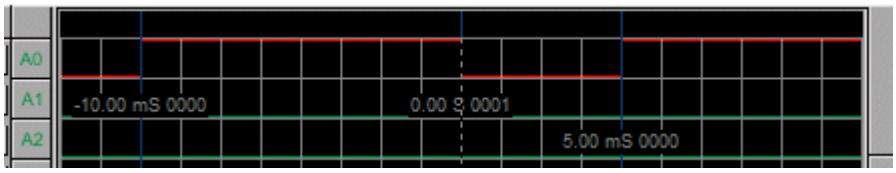


Рисунок 1.29 - Налаштування кольору для сигналу

Після встановлення курсорів необхідно отримати файл для друку. Для цього з контекстного меню (рисунок 1.26) вибираємо Print ... У вікні, що з'явилося вибираємо Microsoft Print to PDF, ставимо галочку навпроти пункту «Друк у файл» і натискаємо кнопку Друк. Стандартно задаємо ім'я файлу та його розташування, зберігаємо файл і після відкриття вирізаємо потрібний фрагмент для звіту (рисунок 1.30).

Рисунок 1.30 - Фрагмент збереженого файлу для звіту

Для отримання тривалості сигналу обчислюємо різницю між двома маркерами. Так тривалість низького сигналу дорівнює 95 - 90 = 5 мілісекунд, а високого 90 - 80 = 10 мілісекунд, що повністю відповідає виставленим значенням у програмі.

Вкажемо ще один спосіб отримання часу між досліджуваними моментами. Для цього у вікні логічного аналізатору (рисунок 1.27) при встановленні міток потрібно підвести курсор до початку досліджуваного моменту та натиснути ліву кнопку вказівника миші. Потім не відпускаючи кнопку довести вказівник миші до кінця досліджуваного моменту і відпустити кнопку.

Встановлення зовнішнього компілятора

Іноді вбудований компілятор Arduino IDE не працює належним чином. У цьому випадку може допомогти використання зовнішнього компілятору. Для цього перейдіть на вкладку Source Code, а потім виберіть пункт меню System -> Compilers Configuration

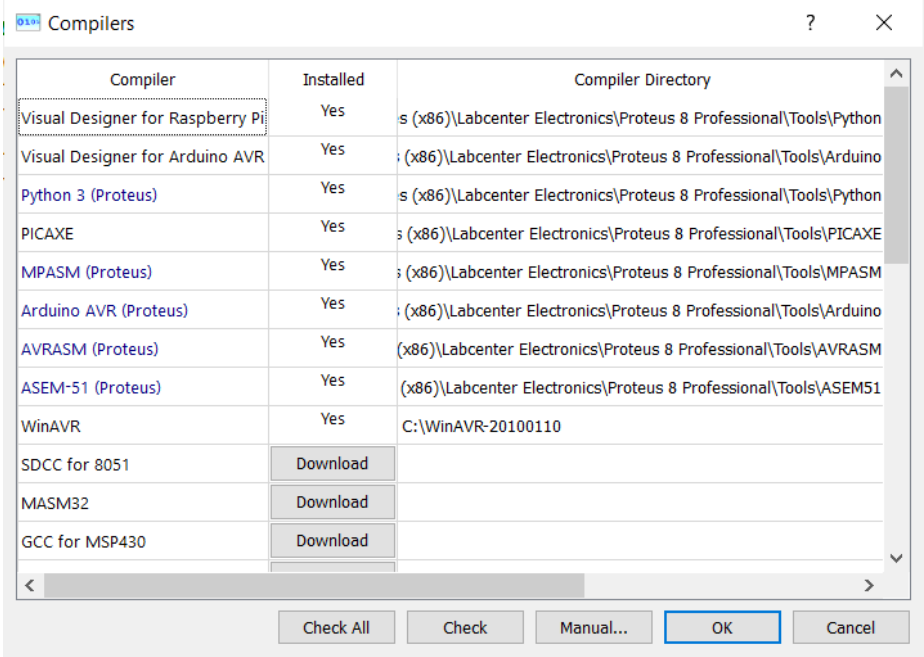


Рисунок 1.31 - Встановлення зовнішнього компілятора

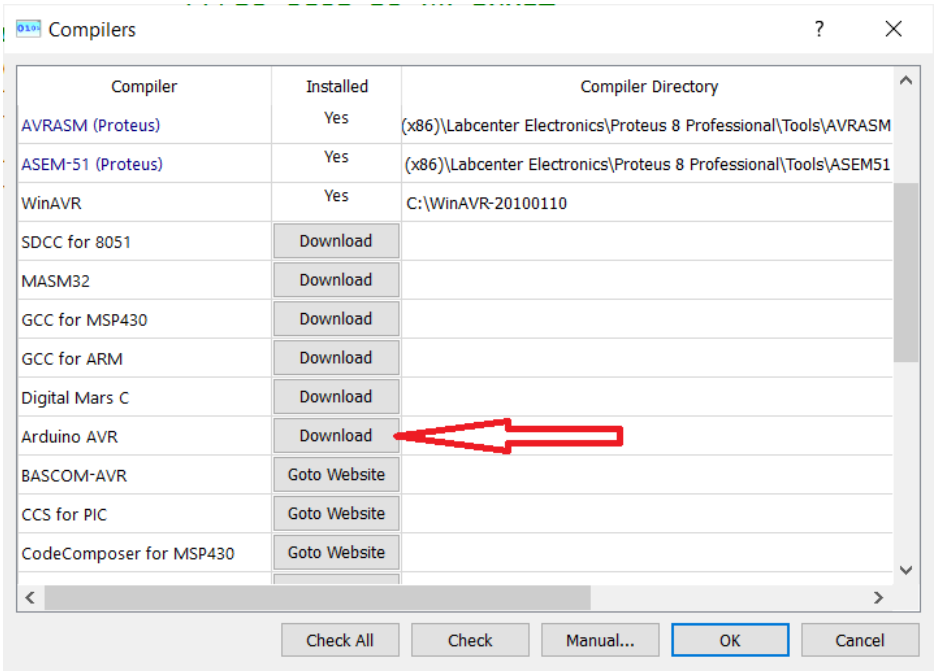


Рисунок 1.32 - Встановлення компілятора для Arduino AVR

Прокрутить стрічку до пункту, показаного червоною стрілкою (рисунок 1.32), натисніть Download і дочекайтеся поки цей рядок зміниться на такий який продемонстровано на рисунку 1.33.

Рисунок 1.33 - Компілятор для Arduino AVR встановлено

Після цього залишиться лише його вибрати та натиснути OK.

Робота з «чистим каменем»

Розглянемо інший варіант роботи з мікроконтролерами Proteus. Повернемося до початку роботи (рисунок 1.13) та виберемо пункт New Project. З'являється вікно, зображене на рисунку 1.34.

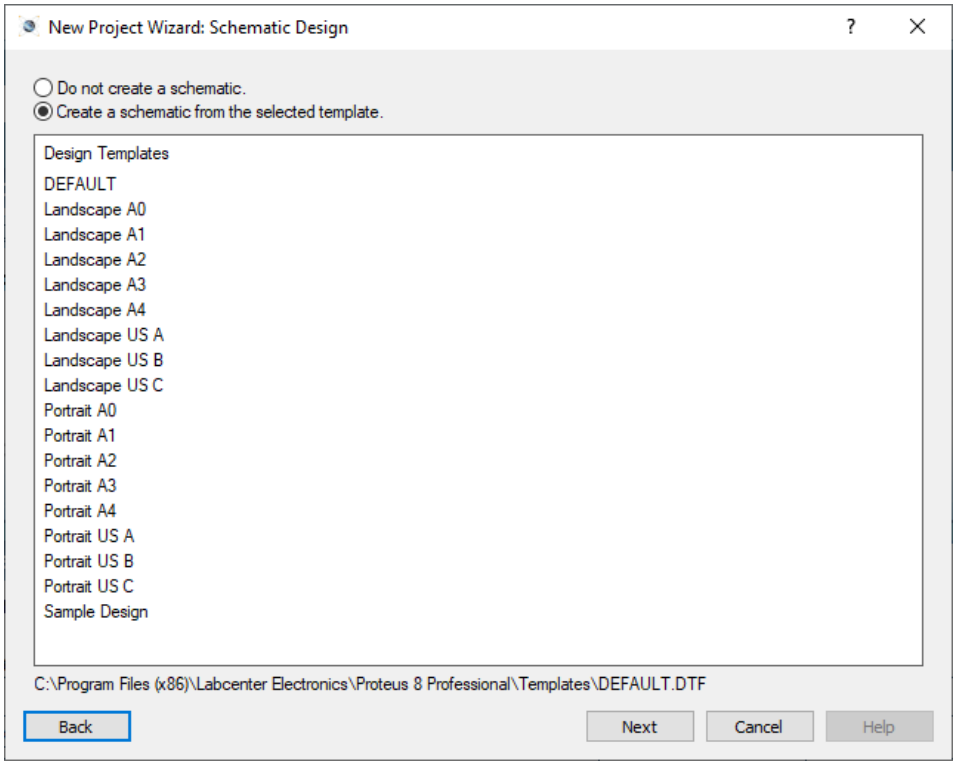


Рисунок 1.34 - Початок роботи з New Project

У ньому обираємо стандартні налаштування: Create a schematic from the selected template, а також DEFAULT. Натискаємо кнопку Next отримуємо вікно зображене на рисунку 1.35.

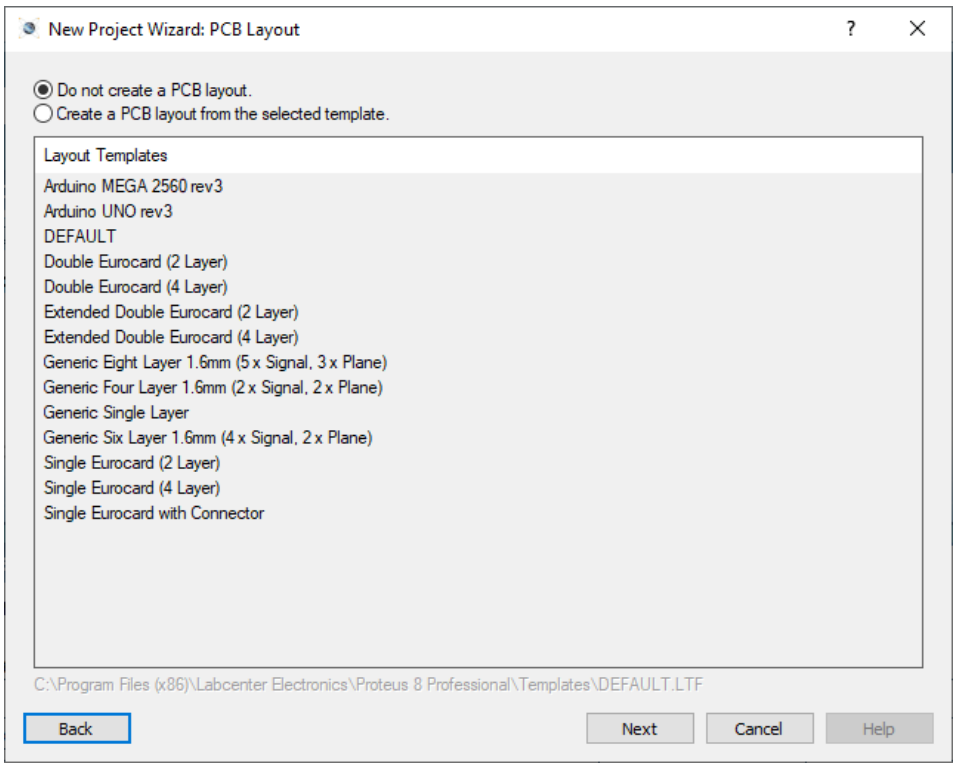


Рисунок 1.35 - Вибір макету (Layout)

Після натискання кнопки Next з'являться вікно (рисунок 1.36).

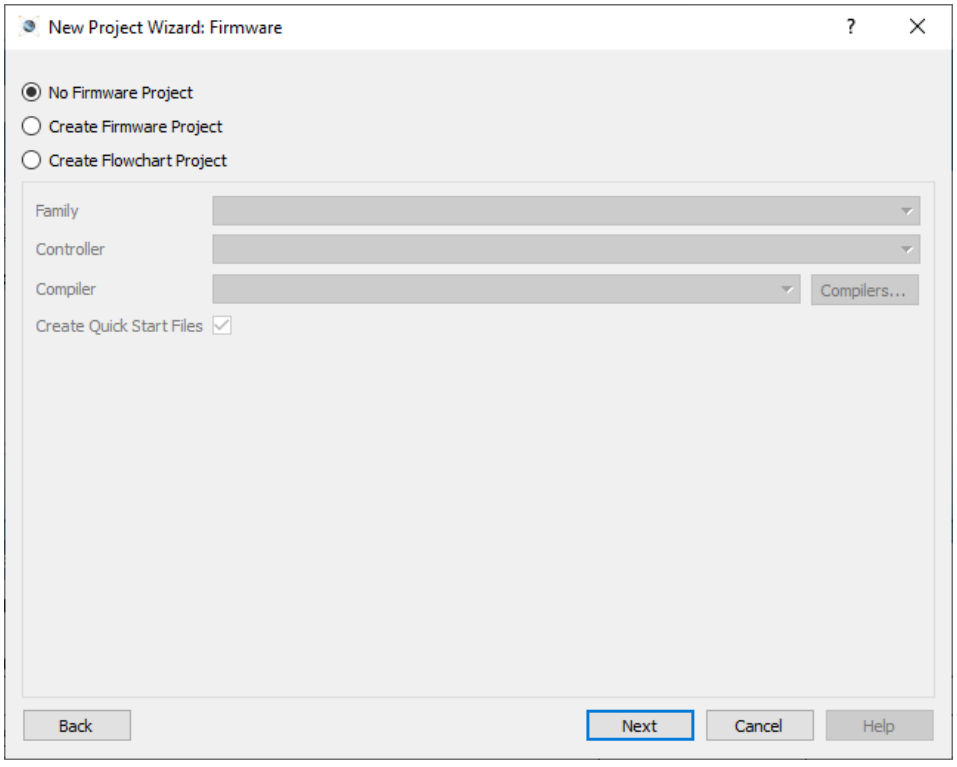


Рисунок 1.36 - Вибір програмного середовища (Firmware)

Залишаємо вибір за замовчуванням та натискаємо Next. Отримуємо наступне вікно (рисунок 1.37)

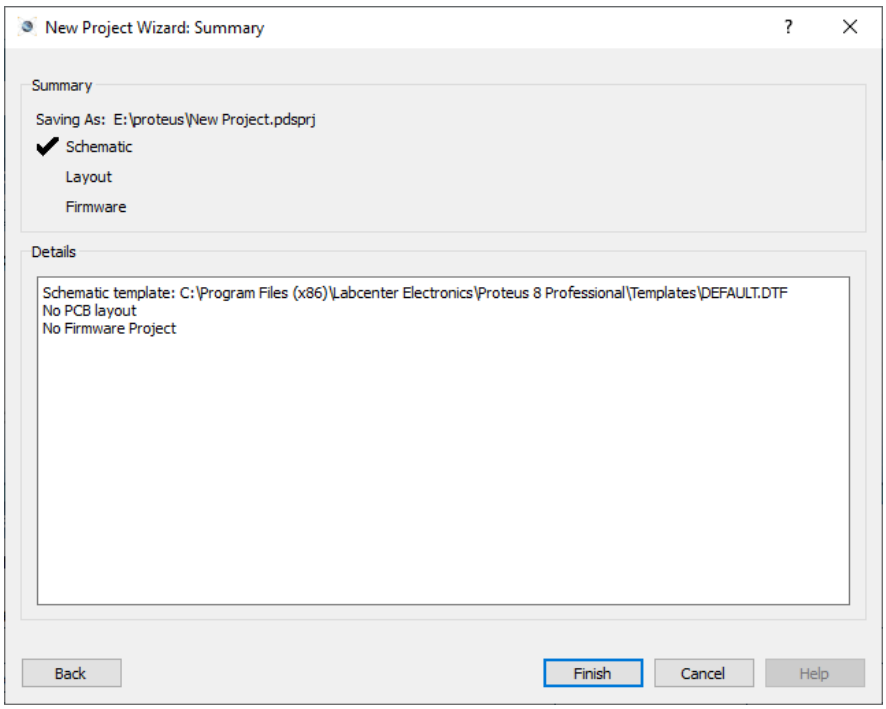


Рисунок 1.37 - Вибір програмного середовища (Firmware)

Обираємо Schematic, та натискаємо кнопку Finish. Після цього ми отримуємо *заготівку* для створення нового проекту (рисунок 1.38).

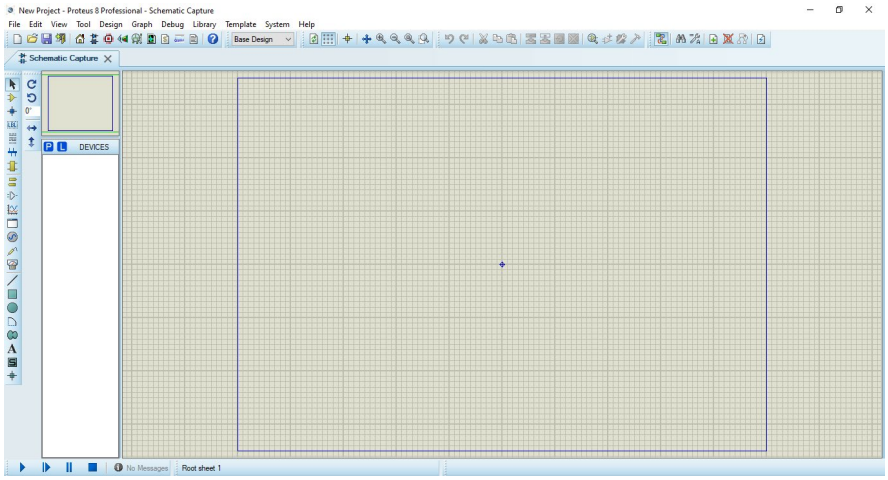


Рисунок 1.38 - Заготівка для нового проекту

Далі встановлюємо необхідні компоненти. Для цього натискаємо кнопку P (Pick Devices) дивись рисунок 1.39. Якщо у вас вибрано іншу вкладку, то спочатку натисніть кнопку Component Mode це жовтенький трикутник зліва, також дивись рисунок 1.39.

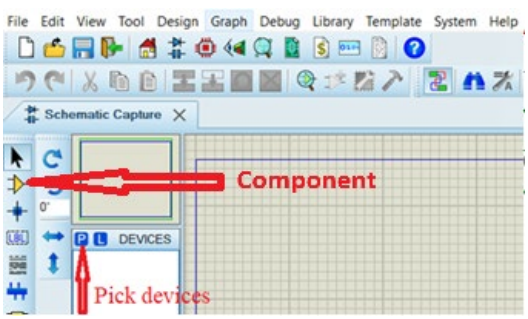


Рисунок 1.39 - Інструменти додавання компонентів

Після виконання цих дій з'являється вікно з вибором множини елементів, розташованих у відповідних групах (рисунок 1.40).

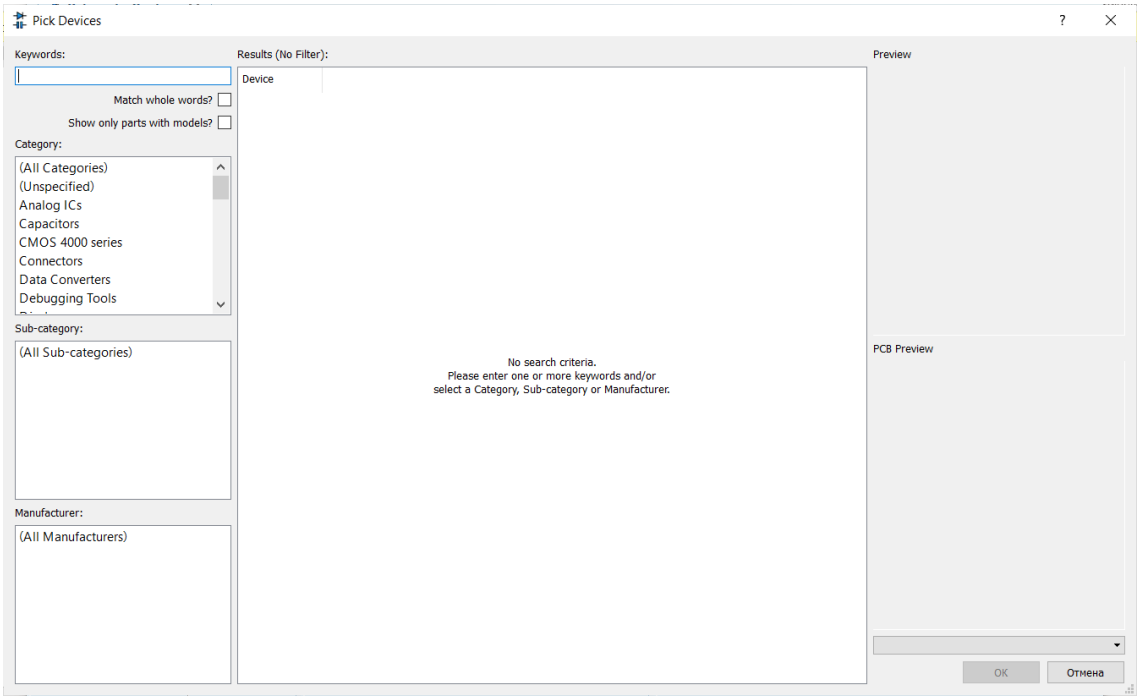


Рисунок 1.40 - Вікно для вибору елементів

Дістатись до потрібного елемента найпростіше, набравши в полі Keywords потрібну назву. Знайдемо мікроконтролер Arduino Mega, для цього напишемо в полі ATmega2560. Отримуємо вікно наступного виду (рисунок 1.41).

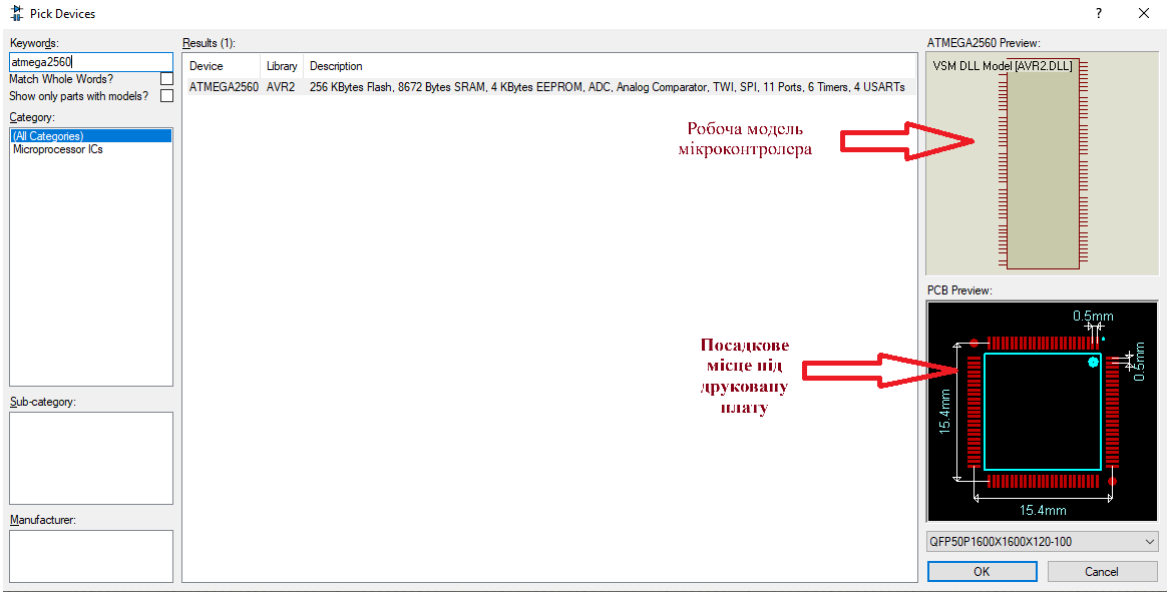


Рисунок 1.41 - Вибір мікроконтролера ATmega2560

З рисунку 1.41 бачимо, що в Proteus присутня модель цього мікроконтролера, також є анімаційна модель і посадкове місце для розробки друкованих плат. Робимо подвійне натискання лівою кнопкою миші на вибраному елементі і бачимо, що він з'являється в полі Devises (рисунок 1.42). При цьому вікно вибору елементів залишається відкритим. Тепер знайдемо анімаційну модель світлодіоду. Вкажемо в полі Keywords, що ми шукаємо активну модель, написавши led active бачимо результат пошуку на рисунку 1.43.

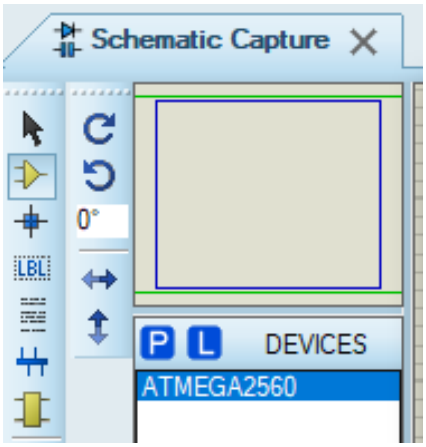


Рисунок 1.42 - Доданий елемент у полі DEVICES

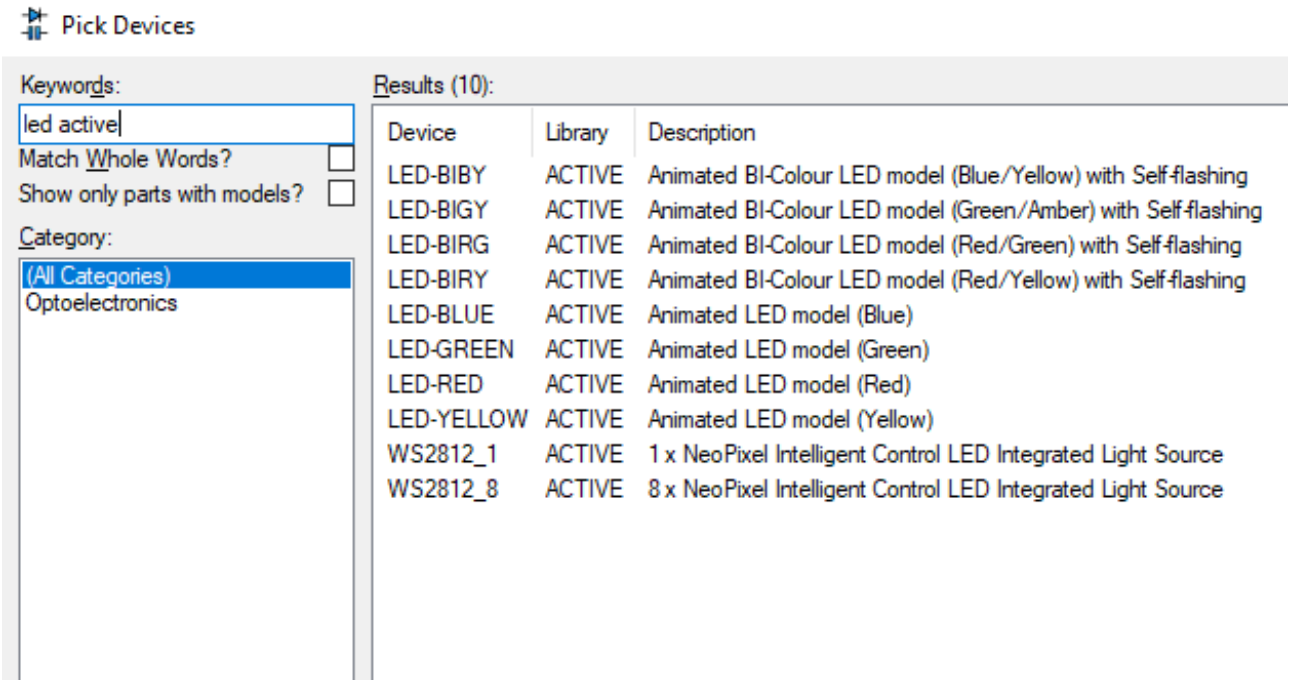


Рисунок 1.43 - Вибір моделі світлодіоду

Виберемо одноколірний світлодіод, наприклад, LED-GREEN. Робимо подвійне натискання на ньому. У полі Devises додався світлодіод. Щоб швидко знайти резистор, напишемо у полі Keywords два слова res device і побачимо наступне вікно результату (рисунок 1.44).

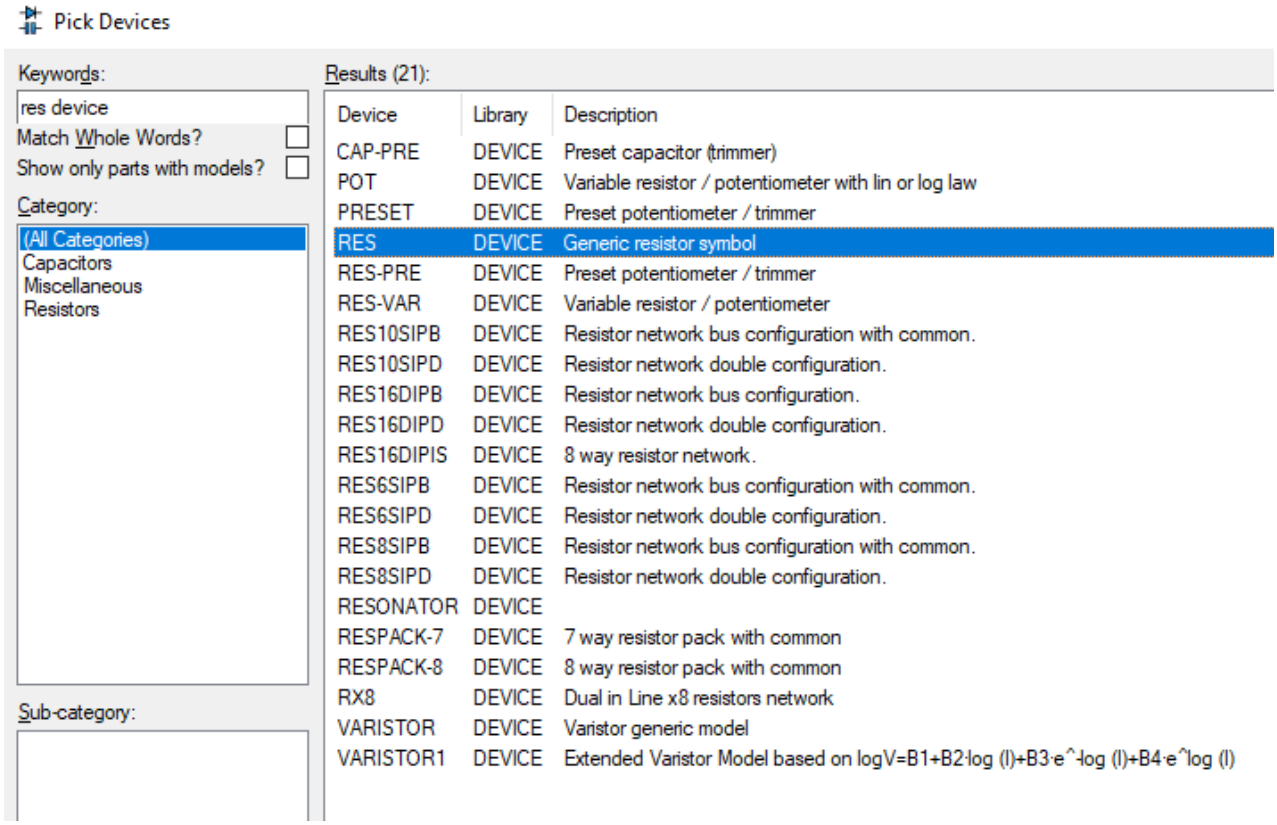


Рисунок 1.44 - Вибір моделі резистора

З отриманого списку виберемо той, який вказаний на рисунку. Його можна виділити та натиснути кнопку ОК, при цьому вибраний елемент з'явиться у полі Devices, а вікно Pick Devices закриється.

Після того, як ми знайшли всі елементи, їх розміщуватимемо в робочому полі Schematic Capture. Для цього клацаємо лівою кнопкою миші по елементу, з яким хочемо працювати, встановлюємо покажчик миші у потрібному місці та ще раз клацаємо лівою кнопкою. Якщо не сподобалося місце установки, елемент можна взяти лівою кнопкою миші і перетягнути. Встановимо, для початку, таким чином мікроконтролер. Налаштуємо середовище розробки для програмування мікроконтролера. Для цього натискаємо правою кнопкою миші по встановленому мікроконтролеру і вибираємо пункт Edit Source Code (рисунок 1.45).

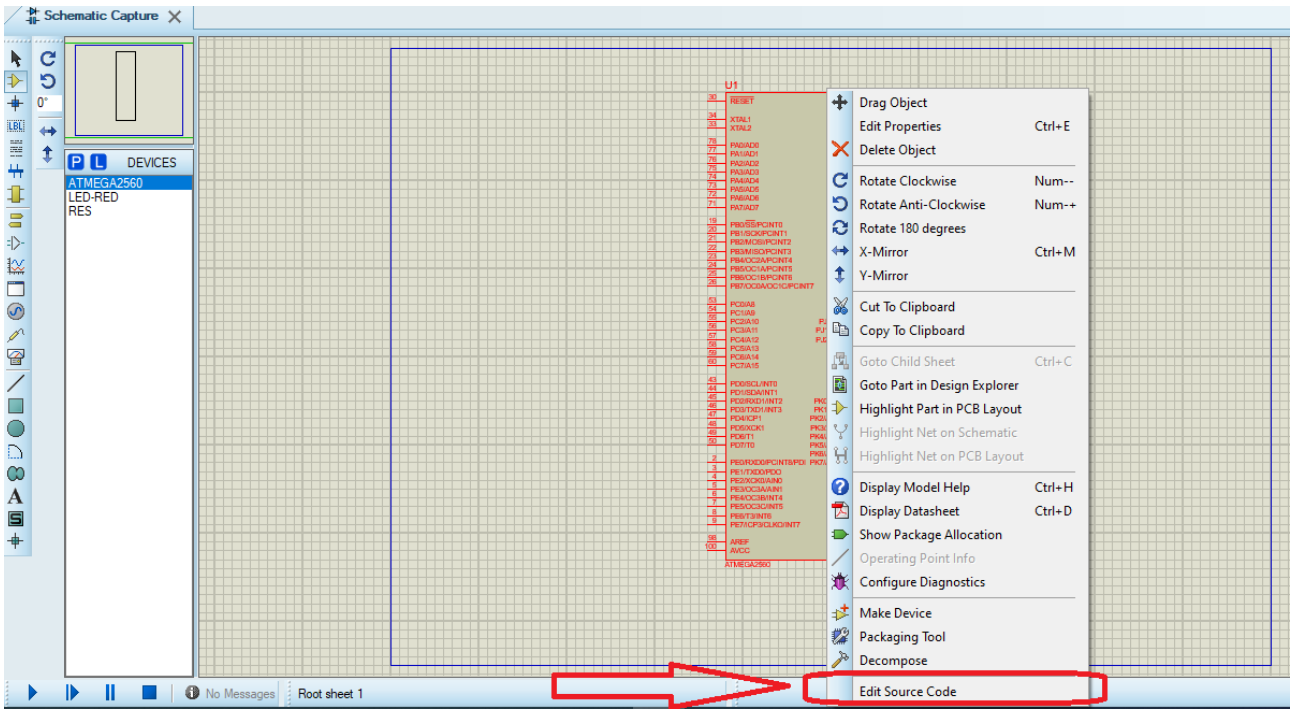


Рисунок 1.45 - Властивості мікроконтролера

Отримуємо наступне вікно (рисунок 1.46).

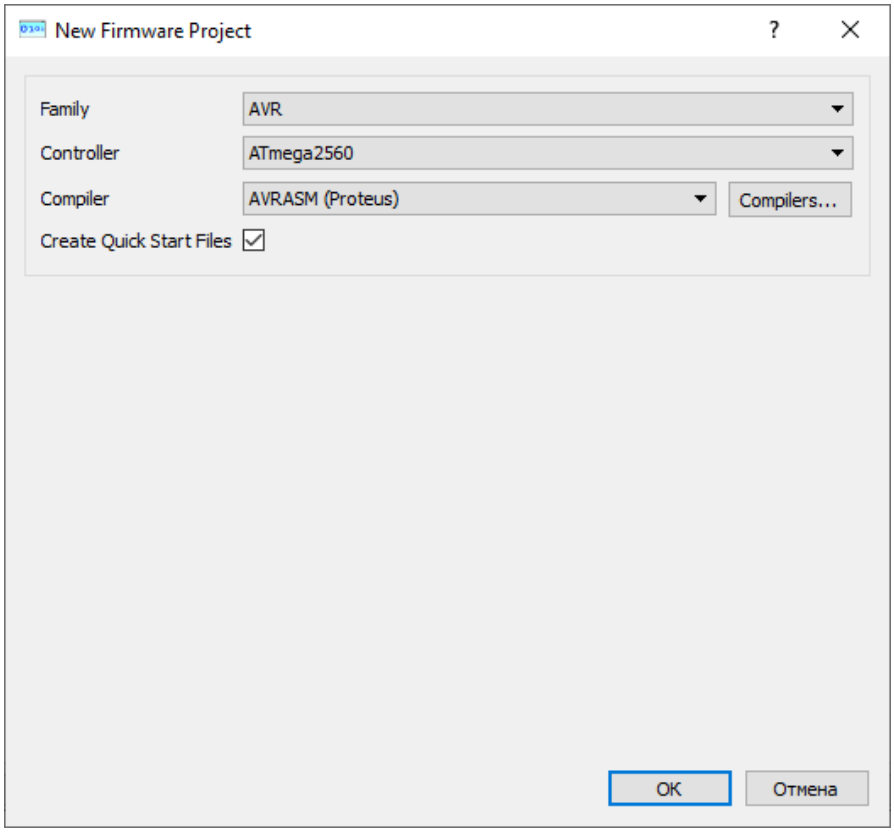


Рисунок 1.46 - Встановлення програмного середовища

У полі Compiler потрібно встановити компілятор, з яким ми будемо працювати. Розкриємо перелік можливих варіантів (рисунок 1.47).

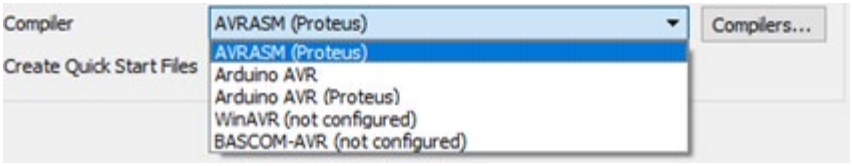


Рисунок 1.47 - Можливі варіанти Firmware

Якщо ви збираєтеся писати програми на асемблері, тоді обирайте AVRASM, для нашого курсу найкраще підійдуть компілятори Arduino AVR (Proteus) або WinAVR. За замовчуванням WinAVR не встановлений на комп'ютер, тому при бажанні його використовувати потрібно натиснути на кнопку Compilers… (рисунок 1.47), потім обрати WinAVR і натиснути Download (рисунок 1.48).

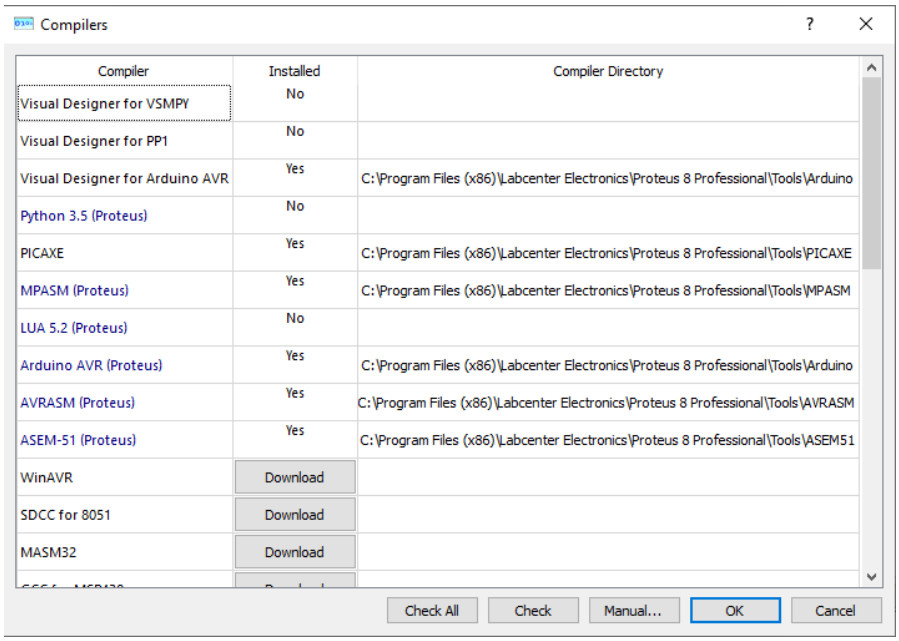


Рисунок 1.48 - Можливі варіанти Firmware

Після завантаження WinAVR він стане доступним для використання (рисунок1.49).

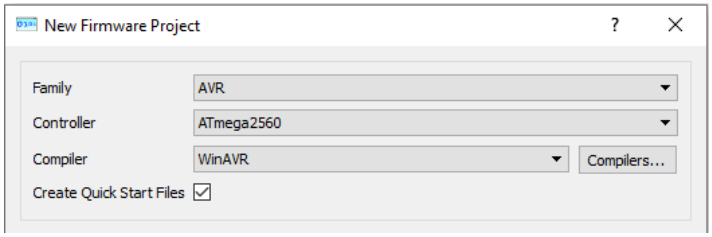


Рисунок 1.49 - Встановлення WinAVR

Натискаємо кнопку OK і у робочому просторі Proteus з'являється вкладка Source Code (рисунок 1.50) з автоматично створеним шаблоном програми.

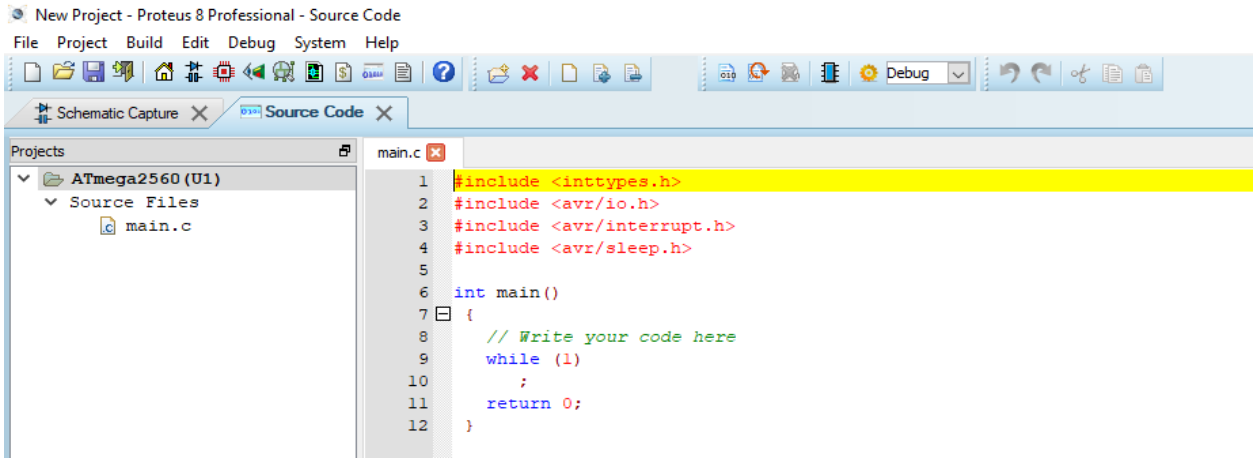


Рисунок 1.50 - Вкладка Source Code

Перенесемо сюди програму **Prog1.3** і змінимо затримку на 8-му рядку на 1000, а на 10-му на 500. Якщо ми зараз увімкнемо анімацію проекту, то все має зібратися без помилок, проект запуститься, проте час включення та вимкнення піна PB7 не буде бажаним . Швидше за все логічна одиниця на PB7 буде 16 секунд (замість однієї), а логічний нуль – 8 секунд (замість 0,5). Це можна побачити по червоному (логічна одиниця) та синьому (логічний нуль) квадратику біля відповідного піна (рисунок 1.51).

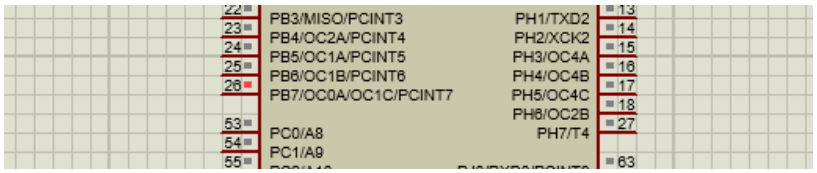


Рисунок 1.51 - Робота піна PB7

Так відбувається тому, що мікроконтролер за замовчуванням налаштований на іншу частоту. Зупинимо симуляцію і двічі натиснемо лівою кнопкою миші на мікроконтролері. З'явиться вікно налаштувань, звернемо увагу на частину, наведену на рисунку 1.52

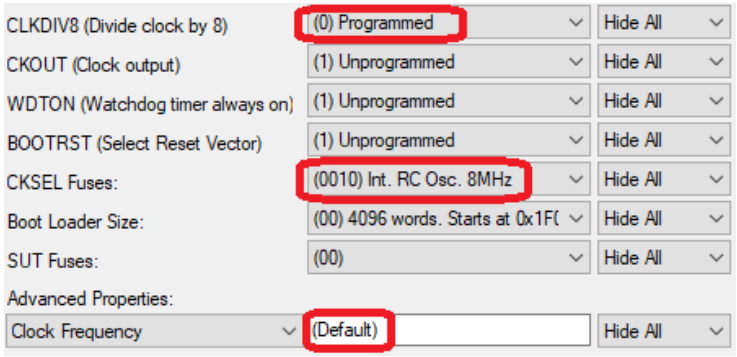


Рисунок 1.52 - Налаштування за замовчуванням

Нам потрібно змінити поля, виділені червоним кольором. Подивимося значення цих полів, встановлених на платформі Arduino Mega (рисунок 1.53) і виберемо їх зі списку, що випадає.

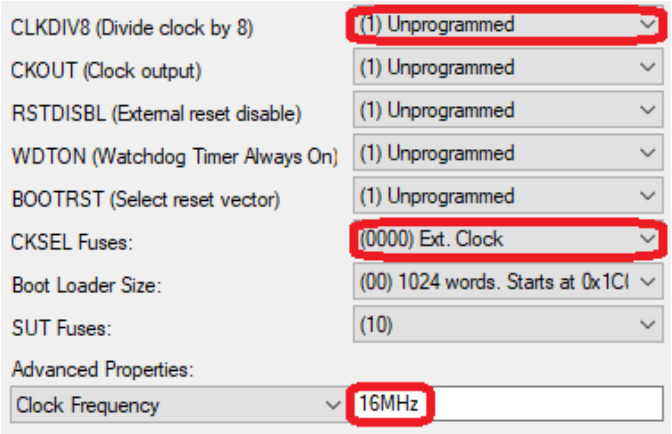


Рисунок 1.53 - Зміна налаштувань

У полі Clock Frequency потрібно ввести значення 16MHz (16000000 Гц). Після цього натискаємо кнопку ОК, вікно закривається і можна працювати над проектом. Після проведених маніпуляцій режим симуляції має видавати правильні часові інтервали.

Тепер підключимо зовнішній світлодіод на пін PB7. Для цього в полі Devices вибираємо світлодіод, а після цього натискаємо лівою кнопкою на робочому полі. Потім вибираємо резистор і встановлюємо його так само в потрібному місці робочого простору. Повинно вийти приблизно як на рисунку 1.54.

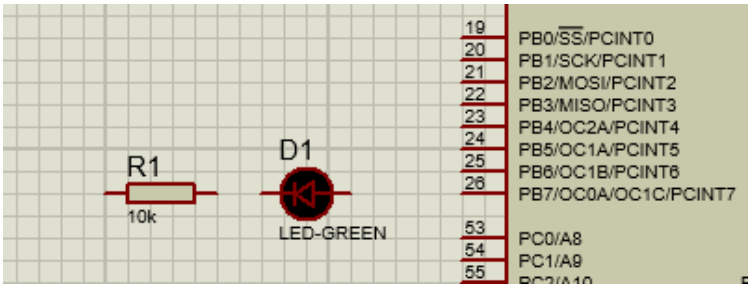


Рисунок 1.54 - Встановлення резистора та світлодіоду до робочого простору

Для повороту вибраного елемента потрібно клацнути на ньому правою кнопкою миші та вибрати з контекстного меню потрібний пункт Rotate… Для обмеження струму через світлодіод використовується резистор номіналом від 200 до 1000 Ом. Для цього двічі клацаємо лівою кнопкою миші на резисторі (саме

на ньому, а не по напису) і в поле Resistance вводимо, наприклад, 300 (рисунок 1.55).

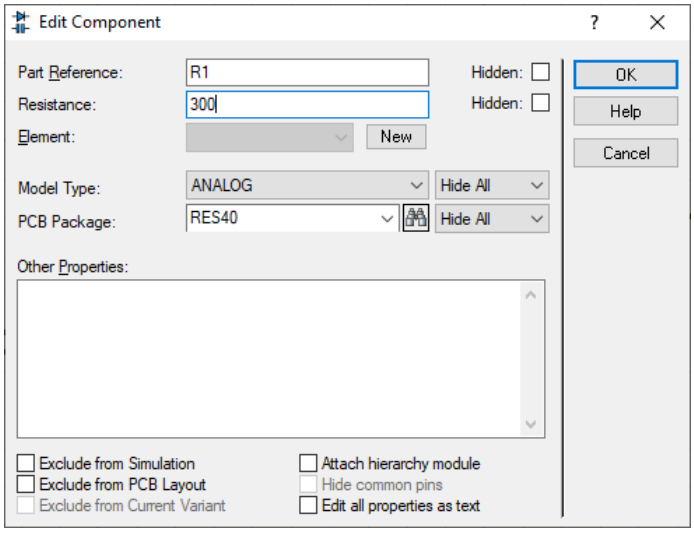


Рисунок 1.55 - Встановлення номіналу резистора

На наступному етапі необхідно встановити землю (Ground) у робочий простір. Для цього натискаємо кнопку Terminals Mode (рисунок 1.56) і в полі TERMINALS знаходимо елемент GROUND. Після цього клацаємо лівою кнопкою в потрібному місці робочого простору на схемі та розміщуємо землю.

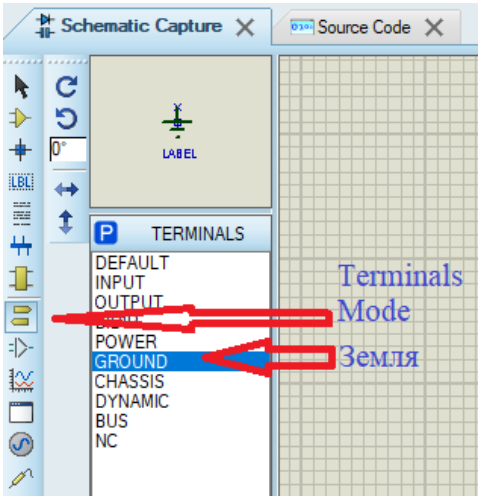


Рисунок 1.56 - Встановлення елемента GROUND

На останньому етапі необхідно з'єднати все це дротами. Для цього підводимо покажчик миші до краю виводу, який з'єднуватимемо, при цьому цей край підсвічується червоним квадратиком. Клацаємо лівою кнопкою миші та ведемо її до іншого виводу, з яким потрібно встановити з'єднання. Як тільки потрібний вивід підсвічується червоним квадратиком, відразу натискаємо на ліву кнопку миші ще раз, з'єднання встановлено. Коли схема буде зібрана (рисунок 1.57), можна перевірити її в дії. Запускаємо анімацію і бачимо, що світлодіод

одну секунду світиться зеленим і півсекунди не світиться.

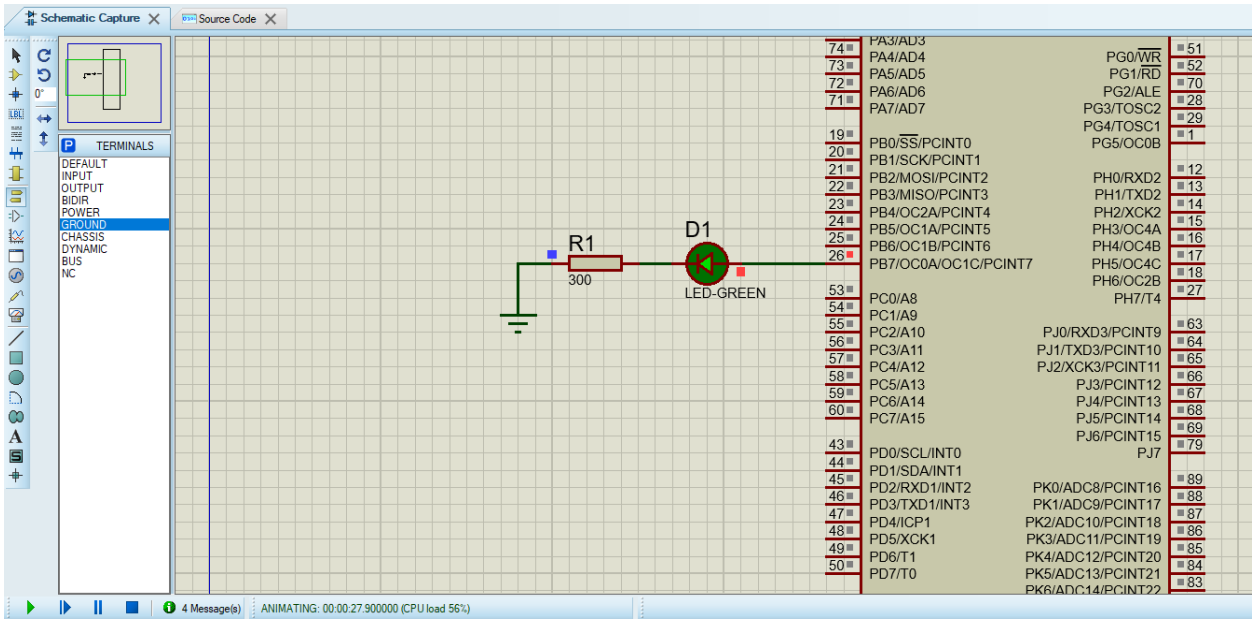


Рисунок 1.57 - Підключення зовнішнього світлодіоду

Для того щоб переконатися в тому, що частота мікроконтролера налаштована правильно, потрібно підключити логічний аналізатор замість світлодіоду і перевірити тривалість низького та високого сигналів за його допомогою, так як це було зроблено раніше.

У заключенні вступної лабораторної роботи відмітимо, що якби нам було необхідно працювати з мікроконтролером ATmega328 (p), то його налаштування мають виглядати так, як зазначено на рисунку 1.58.

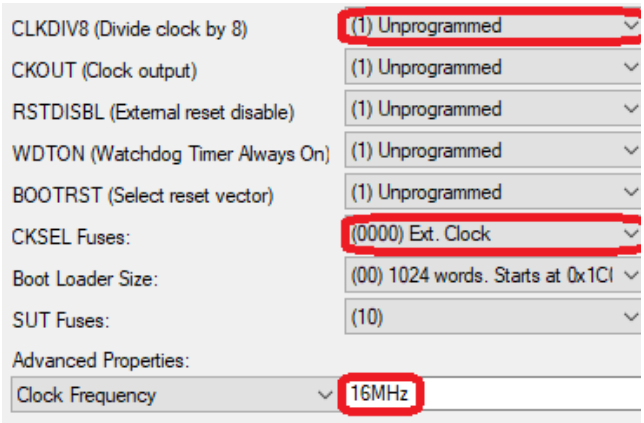


Рисунок 1.58 - Налаштування ATmega328p

Лабораторна робота №2
Керування GPIO

Мета: навчитися налаштовувати GPIO піни мікроконтролерів ATmega328 і ATmega2560 на ВХІД та ВИХІД за допомогою регістрів, а також зчитувати їх стан. Засвоїти принципи роботи з логічним аналізатором.

Теоретичні відомості

Інтерфейс GPIO (*general-purpose input/output*) використовується для зв'язку мікроконтролера з зовнішніми пристроями або іншими мікроконтролерами. В мікроконтролерах ATmega328 та ATmega2560 для позначення пінів які підтримують GPIO використовується позначення P*xn*, де *x* – це назва порту, а *n* – номер біту який відповідає за цей пін. Порт це сукупна назва пінів об'єднаних між собою. Так в мікроконтролері ATmega328 є три порти загального призначення які мають назву B, C, D. Порти B та D мають по вісім пінів, тобто: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7 для порту B та PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7 для порту D. Порт C має шість пінів: PC0, PC1, PC2, PC3, PC4, PC5, PC6. Зв'язок між портами мікроконтролера ATmega328 та платформою Arduino UNO має вигляд:

PB0 = D8, PB1 = D9, PB2 = D10, PB3 = D11, PB4 = D12, PB5 = D13;

PC0 = A0, PC1 = A1, PC2 = A2, PC3 = A3, PC4 = A4, PC5 = A5;

PD0 =D0, PD1 =D1, PD2 =D2, PD3 =D3, PD4 =D4, PD5 =D5, PD6 =D6, PD7 =D7.

Таким чином маємо 20 пінів підтримуючих GPIO на платформі Arduino UNO.

Мікроконтролер ATmega2560 має 11 портів вводу/виводу A, B, C, D, E, F, G, H, J, K, L. Наведемо зв'язок між портами мікроконтролера ATmega2560 та платформою Arduino MEGA:

PA0=D22, PA1=D23, PA2=D24, PA3=D25, PA4=D26, PA5=D27, PA6=D28, PA7=D29;

PB0=D53, PB1=D52, PB2=D51, PB3=D50, PB4=D10, PB5=D11, PB6=D12, PB7=D13;

PC0=D37, PC1=D36, PC2=D35, PC3=D34, PC4=D33, PC5=D32, PC6=D31, PC7=D30;

PD0 =D21, PD1=D20, PD2=D19, PD3=D18, PD4=–, PD5=–, PD6=–, PD7=D38;

PE0=D0, PE1=D1, PE2=–, PE3=D5, PE4=D2, PE5=D3, PE6=–, PE7=–;

PF0=A0, PF1=A1, PF2=A2, PF3=A3, PF4=A4, PF5=A5, PF6=A6, PF7=A7;

PG0=D41, PG1=D40, PG2=D39, PG3=–, PG4=–, PG5=D4, PG6=–, PG7=–;

PH0 =D17, PH1=D16, PH2=–, PH3=D6, PH4=D7, PH5=D8, PH6=D9, PH7=–;

PJ0 =D15, PJ1=D14, PJ2=–, PJ3=–, PJ4=–, PJ5=–, PJ6=–, PJ7=–;

PK0=A8, PK1=A9, PK2=A10, PK3=A11, PK4=A12, PK5=A13, PK6=A14, PK7=A15;

PL0 =D49, PL1=D48, PL2=D47, PL3=D46, PL4=D45, PL5=D44, PL6=D43, PL7=D42.

Таким чином маємо по вісім пінів на портах A, B, C, F, K, L, по п'ять пінів на портах D, E, чотири на G, шість на H та два на J портах які підтримують GPIO інтерфейс на платформі Arduino MEGA. Усього разом отримуємо 54 цифрових та 16 аналогових пінів.

Для керування довільним піном P*xn*, існують три регістра PIN*x*, DDR*x*, PORT*x*. Наприклад, якщо ми хочемо керувати PK4 (четвертим піном порта K). Для цього нам потрібно використовувати четвертий біт регістрів PINK, DDRK, PORTK. Згідно з документації на мікроконтролери четвертий біт регістра PINK має назву PINK4, регістра DDRK – DDK4, регістра PORTK – PORTK4. Взагалі пін P*xn* керується трьома бітами PIN*xn*, DD*xn*, PORT*xn* відповідних регістрів.

За допомогою регістра DDR*x* визначається напрямок роботи піна порту *x*. Тобто якщо біт DD*xn* установлено в «1», то відповідний пін P*xn* налаштовується як ВИХІД. Якщо стан біту DD*xn* скинуто в «0», то пін P*xn* буде працювати як ВХІД.

Налаштування регістру можна виконати двома засобами. Перший засіб годиться якщо треба налаштувати всі піни порту відразу. Так будь-яка із записів

```
DDRK = 0b00010000; // двійкова система числення
DDRK = 0x10;       // шістнадцяткова система числення
DDRK = 16;         // десяткова система числення
```

налаштує четвертий пін порту K, тобто PK4 як ВИХІД, а всі інші піни цього порту як ВХОДИ.

Другий засіб використовується якщо треба встановити один або декілька пінів як ВИХІД (ВХІД), а тип решти пінів залишити не змінним. Будь-яка з записів

```
DDRK |= (1<<DDK4);  // звернення до біту за назвою
DDRK |= (1<<4);     // звернення до біту за номером
```

налаштує пін РК4 як ВИХІД, при чому всі інші піни цього порту не змінять свій тип.

Будь-яка з команд

```
DDRK &= ~(1<<DDK4);   //звернення до біту за назвою
DDRK &= ~(1<<4);      //звернення до біту за номером
```

встановить пін РК4 як ВХІД, при чому всі інші піни цього порту не змінять свій тип.

Зробимо декілька пояснень до коду. Вертикальна риска | (пайп) означає побітову операцію логічного АБО (диз'юнкція - логічне додавання). Знак & (амперсанд) означає побітову операцію логічного І (кон'юнкція - логічне множення). Знак ~ (тильда) означає побітову операцію логічного НЕ (інверсія). Знак << (двійна кутова дужка) означає побітовий зсув ліворуч.

Регістр PORT*x* керує станом пінів.

Коли пін P*xn* налаштований як ВИХІД, то його стан відповідає значенню біта PORT*xn*. Якщо PORT*xn* = 1, то на піні P*xn* знаходиться логічна одиниця (тобто 5 вольт), якщо PORT*xn* = 0, то на піні P*xn* знаходиться логічний нуль (тобто 0 вольт або «земля»).

Коли пін P*xn* налаштований як ВХІД, то якщо PORT*xn* = 0, то він знаходиться в високоімпедансному – Z стані, по суті пін «підвішене у повітрі». Якщо PORT*xn* = 1, то пін знаходиться в INPUT_PULLUP стані, тобто пін «підтягнутий» до плюс 5 вольт, через внутрішній резистор, який здебільше дорівнює 100 кілоом = 10^5 Ом.

Встановлення або скидання біта PORT*xn* відбувається аналогічними командами:

```
PORTK |= (1<<PORTK4);    //Встановлення біта
PORTK &= ~(1<<PORTK4);   //скидання біта
```

Регістр PIN*x* – це регістр читання.

У цьому регістрі знаходиться інформація про рівень напруги який присутній на теперішній момент часу (незалежно від того на який тип ВХІД або ВИХІД налаштовано пін) на всіх пінах порту *x*. Якщо на піні РК4 присутній рівень 5

вольт, то стан біта PINK4 дорівнює логічній «1», якщо – 0 вольт, то – логічний «0».

Приведемо конструкцію яка зчитує стан піна PK4

```
if (PINK&(1<<PINK4)){
      // на піні PK4 логічна одиниця
    }
else{
      // на піні PK4 логічний нуль
    }
```

В методичних цілях приведемо програму яка тримає стан піна **PL3=D46** мікроконтролера ATmega2560 у високому рівні одну мілісекунду, а у низькому – 750 мікросекунд.

Prog2.1

```
1    #define F_CPU 16000000UL    // Частота - 16MHz
2    #include <avr/io.h>         // Бібліотека вводу/виводу
3    #include <util/delay.h>     // Бібліотека затримок
4    int main(void) {
5     DDRL |= (1<<DDL3);      //Налаштовуємо пін PL3=D46 mega2560 на ВИХІД
6     while(1) {
7        PORTL |= (1<<PORTL3);      // Виставляємо лог.1 на PL3
8        _delay_ms(1);              // Затримка 1 мілісекунду
9        PORTL &= ~(1<<PORTL3);     // Виставляємо лог.0 на PL3
10       _delay_us(750);            // Затримка 750 мікросекунд
11    }
12   }
```

Перед завантаженням програми треба в середовищі Arduino IDE вибрати в пунктах меню Tools відповідну платформу (ми працюємо або з Arduino MEGA 2560 або з Arduino UNO), а також порт до якого підключена плата (рисунок 2.1).

Підключимо канал CH1 логічного аналізатору до піна PL3, після завантаження **Prog2.1** на мікроконтролер ATmega2560 та запуску захоплення сигналу логічним аналізатором отримуємо наступні данні (рисунок 2.2). Змінити масштаб можливо натискаючи «+» або «-». Щоб побачити основну інформацію про данні достатньо підвести до необхідної частини діаграми покажчик миші. Для того щоб виміряти час між будь якими подіями треба скористатися пунктами меню Measure -> Add Pair або натиснути Ctrl+T.

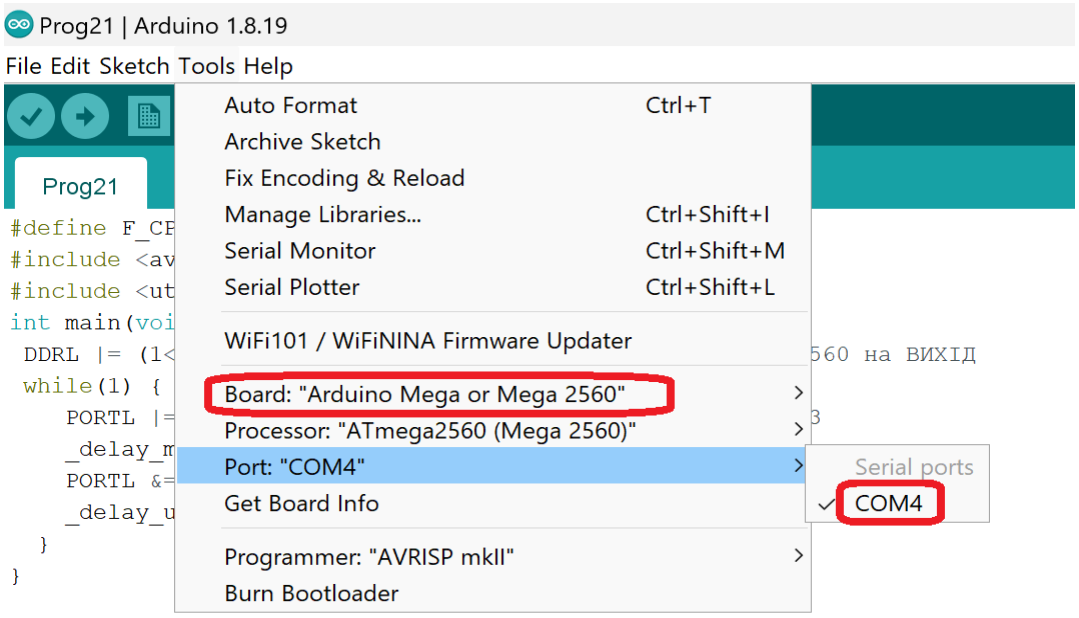
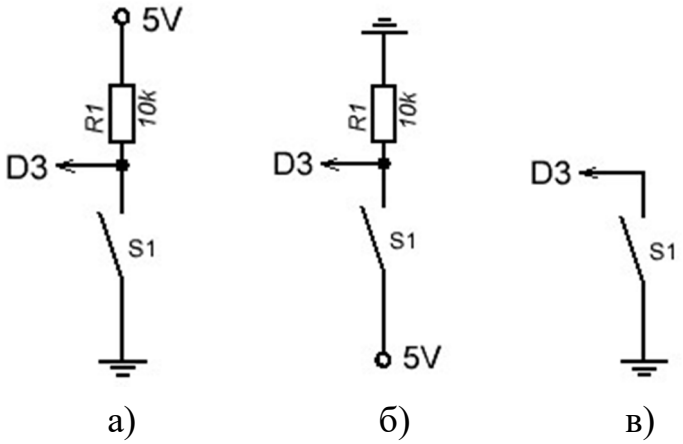


Рисунок 2.1 – Середовище Arduino IDE, вибір плати та COM порту.

Рисунок 2.2 – Отримані данні після завантаження **Prog2.1**

Підключення кнопки



а) б) в)

Рисунок 2.3 – Підключення кнопки: а) з резистором, який підтягує до +5В; б) з резистором, який підтягує до «землі»; в) з внутрішнім резистором.

Розглянемо три основних метода підключення кнопки (рисунок 2.3). Якщо використовується схема підключення а), то на піні D3 буде присутня логічна одиниця коли кнопка не натиснута і логічний нуль – коли натиснута. Якщо будемо використовувати схему б), то на піні D3 буде присутній логічний нуль коли кнопка не натиснута і логічна одиниця – коли натиснута. У випадках а) і б) достатньо налаштувати пін D3 як ВХІД. Наприклад, при застосуванні мікроконтролеру ATmega2560 маємо D3=PE5, тоді частина коду має вигляд:

схема а)

```
DDRE &= ~(1<<DDE5);   // ВХІД
..............
if (PINE&(1<<PINE5)){
      //кнопка не натиснута
    }
else{
      // кнопка натиснута
    }
```

схема б)

```
DDRE &= ~(1<<DDE5);   // ВХІД
..............
if (PINE&(1<<PINE5)){
      //кнопка натиснута
    }
else{
      // кнопка не натиснута
    }
```

Зробимо кілька пояснень до цього коду. Команда

```
PINE&(1<<PINE5)
```

дозволить зрозуміти, який рівень сигналу (високий або низький) знаходиться на піні PE5. Розберемо її докладно. Операція 1 << PINE5 зрушить одиницю ліворуч на п’яту позицію (рахуючи від нуля) і ми отримаємо наступне двійкове число 0b00100000, потім застосуємо побітову операцію логічного І з восьмирозрядним регістром PINE

```
  0b--x-----
& 0b00100000
  ----------
  0b00x00000
```

Нас цікавить стан тільки п’ятого біта регістра PINE, який ми позначили як x, значення всіх інших біт нам неважливо (тому стоять прочерки). Після застосування побітового & у відповіді отримаємо двійкове число 100000 або 0 (нулі на старших розрядах опущені). Таким чином якщо на піні присутній логічний нуль, то ми отримаємо число 0, якщо - логічна одиниця, то отримуємо деяке число більше нуля. Слово деякий означає, що якби ми зчитували другий біт, то отримали б 100, якби шостий, то - 1000000 і тому подібні числа. У мові С число 0 визначається як логічне FALSE, а будь-яке інше число як TRUE.

Якщо використовується схема підключення в), то обов’язково необхідно підключати внутрішній (для мікроконтролеру) резистор який підтягне пін до «+»

живлення. Як було сказано раніше це робиться двома командами, спочатку ми встановлюємо пін як ВХІД за допомогою регістра DDR*x*, потім підключаємо резистор за допомогою регістра PORT*x*. Регістр PIN*x* використовується для зчитування стану відповідного піну. Наведемо частину коду для зчитування стану кнопки підключеною до ATmega2560 за схемою в).

```
DDRE &= ~(1<<DDE5);     // ВХІД
PORTE |= (1<<PORTE5);   // Підключення підтягуючого резистора
..............
if (PINE&(1<<PINE5)){
      //кнопка не натиснута
    }
else{
      // кнопка натиснута
    }
```

Зробимо дуже важливе зауваження. Схеми підключення кнопки які зображені на рисунку 2.3 не є «дурнястійким». Тобто, якщо ви підключите кнопку як вказано на цьому рисунку, а в коді налаштуєте пін як ВИХІД, то при натисканні кнопки можливе коротке замикання. Для запобігання цього рекомендується між піном та кнопкою підключати додатковий резистор 200 – 300 Ом.

Для більшого розуміння наведемо повний код програми яка дозволить натисканням кнопки керувати станом вбудованого світлодіоду який підключено до D13. Схему підключення кнопки виберемо на рисунку 2.3 в). Мікроконтролер ATmega2560 тоді D13=PB7.

Prog2.2

```
1   #define F_CPU 16000000UL    // Частота - 16MHz
2   #include <avr/io.h>         // Бібліотека вводу/виводу
3   int main(void) {
4    DDRB |= (1<<DDB7);     //Налаштовуємо пін PB7=D13 mega2560 на ВИХІД
5    DDRE &= ~(1<<DDE5);    // ВХІД - підключається кнопка
6    PORTE |= (1<<PORTE5);  // Підключення підтягуючого резистора
7     while(1) {
8       if (PINE&(1<<PINE5)){
9         PORTB &= ~(1<<PORTB7);     // Виставляємо лог.0 на PB7
10      }
11      else{
12        PORTB |= (1<<PORTB7);      // Виставляємо лог.1 на PB7
13      }
14    }
15  }
```

Хід роботи

Якщо викладач на сказав інше, то завдання 2.1 та 2.2 непарні варіанти виконують на ATmega328, а парні на ATmega2560. При виконанні 2.3 та 2.4 студенти змінюють мікроконтролер.

Завдання 2.1.

Написати програму мовою С(С++) яка циклічно робить такі дії:

1) встановлює високий рівень на піні Px1;
2) потім одразу встановлює високий рівень на піні Px2;
3) затримка на t1 мікросекунд;
4) встановлює низький рівень на піні Px1;
5) потім одразу встановлює низький рівень на піні Px2.

Таблиця 2.1 – Вхідні данні для завдання 2.1

Варіант	Px1	Px2	t1, мкс
1	D2	D8	1
2	D3	D9	2
3	D4	D10	3
4	D5	D11	4
5	D6	D12	5
6	D7	D13	6
7	D8	A0	7
8	D9	A1	8
9	D10	A2	9
10	D11	A3	10
11	A4	D2	20
12	A5	D3	19
13	D12	D4	18
14	D13	D5	17
15	A0	D6	16
16	A1	D7	15
17	A2	D8	14
18	A3	D9	13
19	D2	D10	12
20	D3	D11	11

Завдання 2.2.

За допомогою логічного аналізатора виконати наступне:

1) виміряти часовий інтервал між першим та другим пунктом завдання №1;
2) обчислити абсолютну та відносну точність третього пункту завдання №1;

3) виміряти часовий інтервал між четвертим та п'ятим пунктом завдання №1;
4) виміряти часовий інтервал між п'ятим та першим пунктом завдання №1.

Завдання 2.3.

Виконати наступні дії:

1) під'єднати за допомогою макетної плати дві кнопки до пінів Px1 та Px2 за схемою вказаною в таблиці, під'єднати до пінів Px3, Px4 та Px5 аноди трьох світлодіодів, катоди через резистори номіналом 510 Ом підключити до «землі» мікроконтролера;
2) зобразити принципіальну та монтажну схеми підключення.

Таблиця 2.2 – Вхідні данні для завдання 2.3

Варіант	Px1	Px2	Схема підключення	Px3	Px4	Px5
1	A4	D9	а	D2	D4	D10
2	A5	D10	б	A4	D6	D11
3	D2	D11	в	A5	D9	D12
4	D3	D12	а	D2	D10	D13
5	D4	D13	б	D3	D11	A0
6	D5	A0	в	D4	D12	A1
7	D6	A1	а	D5	D13	A2
8	D7	A2	б	D6	A0	A3
9	D8	A3	в	D7	A1	A4
10	D9	D8	а	D10	A2	A5
11	D10	A4	б	D9	A3	D1
12	D11	A5	в	A0	D8	D7
13	D12	D2	а	D11	A5	D4
14	D13	D3	б	D12	D2	D6
15	A0	D4	в	D13	D3	D9
16	A1	D5	а	A0	D4	D10
17	A2	D6	б	A1	D5	D11
18	A3	D7	в	A2	D6	D12
19	A4	D8	а	A3	D7	D13
20	A5	D9	б	A4	D10	A0

Завдання 2.4.

Написати програму для схеми з завдання №3 яка робить наступне:

1) при натисканні першої кнопки світиться тільки перший світлодіод;
2) при натисканні другої кнопки світиться тільки другий світлодіод;
3) при натисканні обох кнопок світиться тільки третій світлодіод;
4) якщо кнопки не натиснути, то світлодіоди не світяться.

Лабораторна робота №3

8-бітні таймер-лічильники

Мета: навчитися налаштовувати 8-бітні таймер-лічильники мікроконтролерів ATmega328 і ATmega2560 в різні режими за допомогою регістрів, а також теоретично розраховувати та експериментально визначати параметри роботи.

Теоретичні відомості

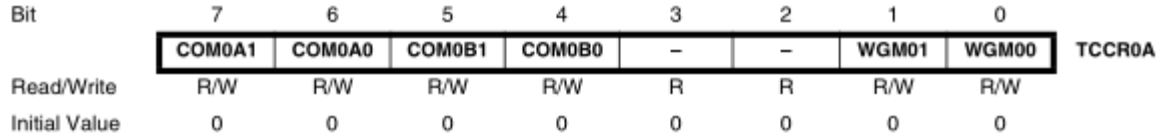
Таймер-лічильники є дуже важливими периферійними пристроями завдяки яким мінімально навантажуючи центральний процесор існує можливість вимірювати час між подіями та підраховувати їх, формувати складні керуючі сигнали, запускати спеціальні підпрограми при виникненні потрібних умов. В мікроконтролерах ATmega328 і ATmega2560 існують два 8-бітних таймер-лічильника – це Timer/Counter0 і Timer/Counter2. Розглянемо регістри найпростішого таймер-лічильника – Timer/Counter0.

Регістр TCNT0 (Timer/Counter Register) підраховує імпульси, тому він має назву – рахунковий регістр. Всі регістри 8-бітних таймер-лічильників 8-бітні, тому в TCNT0 може знаходитись число від 0 до 255. З кожним наступним імпульсом це число може або інкрементуватись (збільшуватись) або декрементуватись (зменшуватись). Регістр TCNT0 доступний як для зчитування так і для запису.

Регістри OCR0A і OCR0B (Output Compare Register A, B) це регістри порівняння, тобто значення які знаходяться в цих регістрах постійно порівнюються з поточним значенням регістру TCNT0. При збігу значень відбуваються відповідні події, які залежать від налаштованого режиму роботи. Регістри OCR0A і OCR0B пов'язані з пінами OC0A і OC0B відповідно. В мікроконтролері ATmega328 OC0A=PD6=D6, OC0B=PD5=D5. В мікроконтролері ATmega2560 OC0A=PB7=D13, OC0B=PG5=D4.

Для налаштування режимів роботи існують регістри TCCR0A і TCCR0B (Timer/Counter Control Register A, B). Налаштування відбувається шляхом встановлення необхідних значень (0 або 1) відповідних бітів.

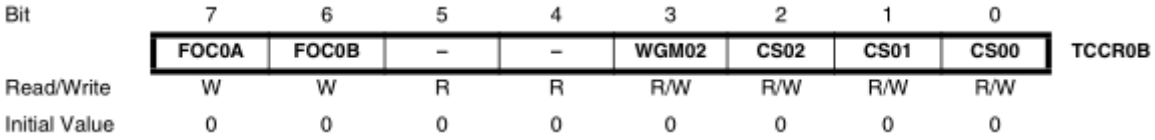
Регістр TCCR0A мікроконтролера ATmega328 описано в пункті 19.9.1 (сторінка 140), а мікроконтролера ATmega2560 на сторінці 129. Регістри цих мікроконтролерів не чим не відрізняються тому зобразимо їх на одному рисунку (рисунок 3.1).



Bit	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.1 - Регістр TCCR0A.

Аналогічно регістр TCCR0B мікроконтролерів ATmega328 і ATmega2560 зображено на сторінках 144 і 132 відповідно. Приведемо його на рисунку 3.2.



Bit	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 3.2 - Регістр TCCR0B.

Розглянемо за що відповідають біти регістрів TCCR0A та TCCR0B. Почнемо з бітів CS00, CS01, CS02 (**Clock Select**) регістра TCCR0B. Ці біти дозволяють вибрати частоту на якій буде працювати таймер-лічильник. У технічній документації на ATmega328 дивись таблицю 19-10, а на ATmega2560 – таблицю 80. Згадана таблиця має вигляд

Таблиця 3.1 – Вибір робочої частоти таймеру

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Додаймо декілька пояснень. Якщо всі біти CS0[0-2] скинуто, тобто вони дорівнюють нулю, то таймер-лічильник зупинене. Якщо тільки один біт CS00 встановлено в одиницю, то таймер-лічильник працює на частоті мікроконтролера (для платформ Arduino UNO та MEGA частота дорівнює

16 МГц). Якщо встановлено тільки біт CS01=1, то частота роботи таймер-лічильника у вісім разів менше, тобто дорівнює 2 МГц і так далі. Якщо обрано дві останні строчки таблиці 3.1, то таймер-лічильник буде працювати на частоті зовнішнього сигналу який поступає на пін T0 = PD4 = D4 (для ATmega328) або T0 = PD7 = D38 (для ATmega2560). Причому у разі передостанньої строчки значення таймер-лічильника буде змінюватись по негативному фронту, а якщо обрано остання строчка, то – по позитивному фронту.

Основні режими роботи налаштовуються за допомогою бітів WGM (**Waveform Generation Mode**), а саме WGM00, WGM01 (регістр TCCR0A) і WGM02 (регістра TCCR0B). Різні режими вказані в таблиці 19-9 мікроконтролера ATmega328 та таблиці 79 мікроконтролера ATmega2560. Приведемо важливу для нас частину цих таблиць

Таблиця 3.2 – Вибір режимів роботи таймеру

№	WGM02	WGM01	WGM00	Режим	TOP
0	0	0	0	Normal	0xFF
1	0	0	1	PWM, Phase Correct	0xFF
2	0	1	0	CTC	OCR0A
3	0	1	1	Fast PWM	0xFF
4	1	0	0	Reserved	–
5	1	0	1	PWM, Phase Correct	OCR0A
6	1	1	0	Reserved	–
7	1	1	1	Fast PWM	OCR0A

Як видно з цієї таблиці існують такі режими:

Normal

CTC - Clear Timer on Compare Match

Fast PWM - Fast Pulse-Width Modulation

PWM, Phase Correct - Pulse-Width Modulation, Phase Correct

Normal (звичайний) - режим, в якому регістр TCNT0 рахує імпульси від нуля до TOP. Згідно таблиці 3.1 верхня границя TOP = 255. Після досягнення цього значення TCNT0 скидається в нуль.

Режим **CTC** – скидання при збігу. В цьому режимі рахунковий регістр TCNT0 також рахує імпульси. Але коли значення TCNT0 співпадає з регістром порівняння OCR0A, то регістр TCNT0 буде скинуто в нуль.

Як будуть вести себе піни OC0A та OC0B залежить від виставлених біт (**Compare Output Mode**) COM0A0, COM0A1 та COM0B0, COM0B1 регістра TCCR0A (рисунок 3.1) та режимів роботи. Варіанти поведінки пінів у режимах NORMAL та CTC при збігу значень регістрів OCR0A та OCR0B з регістром TCNT0 описані в таблиці 3.2 (див. таблиці 19-3, 19-6 технічного опису на мікроконтролер ATmega328 та таблиці 73, 76 технічного опису на мікроконтролер ATmega2560).

Таблиця 3.3 – Поведінка пінів OC0A, OC0B в режимі Normal і CTC

COM0*x*1	COM0*x*0	Опис
0	0	Пін OC0*x* відключено від таймера
0	1	Стан піна OC0*x* змінюється на протилежний
1	0	Пін OC0*x* скидається в «0»
1	1	Пін OC0*x* встановлюється в «1»

Примітка: *x* = A або B.

Для того щоб стан пінів OC0A та OC0B змінювався необхідним чином треба ці піни налаштувати як ВИХІД. Приведемо приклад програми яка встановлює звичайний режим (режим №0) роботи таймер-лічильника 0 на мікроконтролері ATmega328.

Prog3.1

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
6      DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
7      TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
8      TCCR0A |= (1<<COM0A0); //переключення OC0A при збігу OCR0A=TCNT0
9      OCR0A = 127; // максимум 255
10     TCCR0A |= (1<<COM0B0); //переключення OC0B при збігу OCR0B=TCNT0
11     OCR0B = 0; // максимум 255
12       while (1){};        //основна програма
13   }
```

Після завантаження **Prog3.1** в мікроконтролер ATmega328 отримуємо наступну часову діаграму (рисунок 3.3).

Рисунок 3.3 - Часова діаграма після завантаження **Prog3.1**.

Підрахуємо час одного тику таймера

$$t_{тика} = \frac{N}{f_{clk}} = \frac{N}{16 \cdot 10^6} = N \cdot 62{,}5 \cdot 10^{-9} c = N \cdot 62{,}5 нс\,, \qquad (3.1)$$

де N – дільник згідно таблиці 3.1.

Так як у цьому режимі таймер рахує від 0 до 255, то під час одного імпульсу проходить 256 тактів, це означає, що

$$t_{імп} = 256 \cdot t_{тика} = 256 \cdot N \cdot 62{,}5 нс = N \cdot 16 мкс\,. \qquad (3.2)$$

У випадку програми **Prog3.1** маємо N=1024, тому теоретичний розрахунок тривалості імпульсу дорівнює $t_{імп} = 16{,}384 мс$. Що дуже близько до реального значення (рисунок 3.3).

Приведемо приклад програми яка встановлює режим СТС (режим №2 таблиці 3.2) роботи таймер-лічильника 0 на мікроконтролері ATmega328.

Prog3.2

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
6      DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
7      TCCR0A |= (1<<WGM01);  //режим СТС № 2
8      TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
9      TCCR0A |= (1<<COM0A0); //переключення OC0A при збігу OCR0A=TCNT0
10     OCR0A = 200; // максимум 255
11     TCCR0A |= (1<<COM0B0); //переключення OC0B при збігу OCR0B=TCNT0
12     OCR0B = 100; // максимум OCR0A
13
14       while (1){};        //основна програма
15   }
```

Часова діаграма після завантаження наведена на рисунку 3.4.

Рисунок 3.4 - Часова діаграма після завантаження **Prog3.2**.

В цьому випадку частота сигналу на пінах OC0A і OC0B розраховується за формулою

$$f_{OC0x} = \frac{f_{clk}}{2 \cdot N \cdot (OCR0A + 1)}, \tag{3.3}$$

де f_{clk} - частота роботи мікроконтролера, у нашому випадку $f_{clk} = 16 МГц$, N – дільник, *OCR0A* – значення відповідного регістру. З формули (3.3) бачимо, що теоретична частота яку генерує програма **Prog3.2** дорівнює

$$f_{OC0x} = \frac{16 \cdot 10^6}{2 \cdot 1024 \cdot (200 + 1)} = \frac{16 \cdot 10^6}{411648} = 38{,}868 Гц. \tag{3.4}$$

Пояснімо, ще один момент. В режимі №2 частота сигналу залежить тільки від значення регістру *OCR0A*. При цьому, якщо значення регістру *OCR0B* буде не більше *OCR0A*, то на піні буде *OC0B* генеруватися такий саме сигнал як і на піні *OC0A* але між ними буде різниця фаз пропорційна різниці значень між регістрами *OCR0A* і *OCR0B*.

Перейдемо до розгляду режиму **Fast PWM**. Це режим швидкого ШІМ (широтна імпульсна модуляція). Згідно таблиці 3.2 у восьми-бітного таймера існують два режими Fast PWM, це режими під номерами 3 та 7. В режимі швидкого ШІМ лічильник рахує від нуля до TOP. В режимі під номером 3 TOP = 255, а в режимі під номером 7 TOP = OCR0A, тобто до значення яке знаходиться в цьому регістри. Після цього значення рахункового регістра скидається в нуль. Частота ШІМ сигналу в режимах Fast PWM розраховується за формулою

$$f_{OC0x} = \frac{f_{clk}}{N \cdot (1 + TOP)}. \tag{3.5}$$

Поведінка пінів OC0A та OC0B залежить від виставлених бітів (Compare Output Mode) COM0A0, COM0A1 та COM0B0, COM0B1 регістра TCCR0A (рисунок 3.1) та визначається згідно з таблиці 3.4.

Таблиця 3.4 – Поведінка пінів OC0A, OC0B в режимі Fast PWM

COM0*x*1	COM0*x*0	Опис
0	0	Пін OC0*x* відключено від таймера.
0	1	*Для піна OC0A*: якщо WGM02 = 0, пін OC0A відключено від таймера; якщо WGM02 = 1, стан піна OC0A змінюється на протилежний коли значення регістрів TCNT0 і OCR0A співпадають. *Для піна OC0B*: зарезервовано.
1	0	*Режим неінвертованого ШІМ-сигналу* Пін OC0*x* скидається в «0» коли значення регістрів TCNT0 і OCR0*x* співпадають. OC0*x* встановлюється в «1» коли значення TCNT0 = 0x00.
1	1	*Режим інвертованого ШІМ-сигналу* Пін OC0*x* встановлюється в «1» коли значення регістрів TCNT0 і OCR0*x* співпадають. OC0*x* скидається в «0» коли значення TCNT0 = 0x00.

Примітка: *x* = A або B.

Найчастіше використовуються два останніх режими згідно таблиці 3.4.

Для режиму №3 (таблиця 3.2) значення яке знаходиться в порівняльних регістрах OCR0*x* (*x* = A або B) має назву ШІМ параметр. Неінвертований та інвертований ШІМ режими відрізняються тим, що у першому випадку чим більший ШІМ параметр тим більше середнє значення напруги на ШІМ піні, а в другому випадку при збільшенні ШІМ параметру середнє значення напруги – зменшується.

Наведемо приклад з використання режиму №3.

Prog3.3

```
#define F_CPU 16000000UL  //  частота - 16MHz
#include <avr/io.h>       // Бібліотека вводу/виводу

int main(void){
  DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
  DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
  TCCR0A |= (1<<WGM01)|(1<<WGM00);  //режим Fast PWM №3
  TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
  TCCR0A |= (1<<COM0A1);             //неінвертований ШІМ
  OCR0A = 200; // максимум 255
  TCCR0A |= (1<<COM0B1)|(1<<COM0B0); //інвертований ШІМ
  OCR0B = 170; // максимум 255

    while (1){};        //основна програма
}
```

Часова діаграма після завантаження **Prog3.3** наведена на рисунку 3.5.

Рисунок 3.5 - Часова діаграма після завантаження **Prog3.3**.

Розглянемо цей рисунок більш детально. Спочатку розрахуємо частоту ШІМ сигналу по формулі (3.5), отримуємо

$$f_{OC0x} = \frac{16 \cdot 10^6}{1024 \cdot (1+255)} = \frac{16 \cdot 10^6}{262144} = 61{,}035 Гц . \quad (3.6)$$

З рисунка бачимо, що частота на обох каналах одна й те сама та дорівнює 60,726 Гц, що приблизно співпадає з теоретичними розрахунками. На каналі А генерується неінвертований ШІМ сигнал, це зрозуміло по тому, що тривалість високого рівня сигналу t_1 більша за тривалість низького рівня t_2 при ШІМ параметрі рівному 200, тобто більшому ніж TOP/2. Розрахувати тривалості t_1 та t_2 можна по звичайній пропорції. Для цього спочатку розрахуємо період ШІМ сигналу

$$T = \frac{1}{f_{OC0x}} = \frac{1}{61{,}035} = 16{,}38 мс . \quad (3.7)$$

Тоді складемо пропорцію

$$\begin{matrix} 1+TOP \leftrightarrow T \\ 1+PWM \leftrightarrow t_1 \end{matrix}, \quad (3.8)$$

де PWM – ШІМ параметр. Звідси отримуємо

$$t_1 = \frac{(1+200) \cdot 16{,}38}{1+255} = 12{,}86 мс , \quad (3.9)$$

що доволі близько з експериментальним значенням.

Тривалість t_2 розраховується за такою самою пропорцією, але треба пам'ятати, що це тривалість низького рівня, тому що пін OC0B налаштовано на інвертований ШІМ сигнал. Таким чином маємо

$$t_2 = \frac{171 \cdot 16{,}38}{256} = 10{,}94 мс\,. \qquad (3.10)$$

Пояснимо чому для правильного розрахунку використовується така пропорція. Для цього в програмі **Prog3.3** покладемо `OCR0A = OCR0B = 0`. Тоді отримуємо наступну часову діаграму (рисунок 3.6).

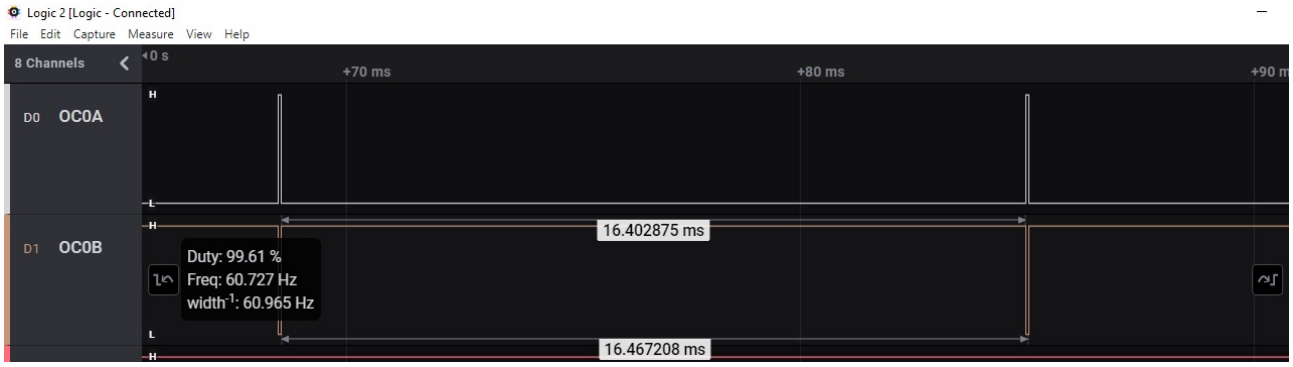


Рисунок 3.6 - Часова діаграма після встановлення `OCR0A=OCR0B=0` в **Prog3.3**.

З рисунку 3.6 бачимо, що при `OCR0A = 0` на піні OC0A генерується не нульовий сигнал, а є ділянки тривалістю 64,33 мкс на яких присутній високій рівень. І навпаки на піні OC0B, який налаштовано на інверсний ШІМ, генерується при `OCR0B = 0` не тільки високий сигнал, а ще є ділянки тривалістю 64,33 мкс низького рівня. Так відбувається тому що при скиданні регістра TCNT0 з 255 до нуля проходить один такт таймеру. Згідно з формули (3.1) час одного такту таймера становить у нашому випадку

$$t_{такт} = N \cdot 62{,}5 нс = 1024 \cdot 62{,}5 нс = 64 мкс\,. \qquad (3.11)$$

З другого боку, якщо в формулі (3.8) покладемо $PWM = 0$, то отримуємо, що час одного такту таймера дорівнює

$$t_1 = \frac{(1+0) \cdot 16{,}38}{256} = 63{,}98 мкс\,, \qquad (3.12)$$

тобто практично стільки ж скільки і по формулі (3.11).

Для режиму №7 Fast PWM (таблиця 3.2) регістр OCR0A визначає частоту ШІМ сигналу, а регістр OCR0B визначає тривалість високого (низького) рівня. Напишемо програму **Prog3.4** яка запускає таймер в цьому режимі та подивимось, що генерується на пінах OC0A, OC0B.

Prog3.4

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
6      DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
7      TCCR0A |= (1<<WGM01)|(1<<WGM00);  //режим Fast PWM №7
8      TCCR0B |= (1<<WGM02);
9
10     TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
11     TCCR0A |= (1<<COM0A1);              //неінвертований ШІМ
12     OCR0A = 200; // максимум 255
13     TCCR0A |= (1<<COM0B1)|(1<<COM0B0); //інвертований ШІМ
14     OCR0B = 170; // максимум OCR0A
15
16       while (1){};        //основна програма
17   }
```

Частота ШІМ сигналу згідно (3.5) дорівнює

$$f_{OC0x} = \frac{f_{clk}}{N \cdot (1 + TOP)} = \frac{16 \cdot 10^6}{1024 \cdot (1 + 200)} = \frac{16 \cdot 10^6}{205824} = 77{,}74 Гц . \qquad (3.11)$$

Налаштування піна OC0A на строках 5, 11 не важливі, так як в цьому режимі цей пін не використовується. Часова діаграма після завантаження **Prog3.4** приведена на рисунку 3.7.

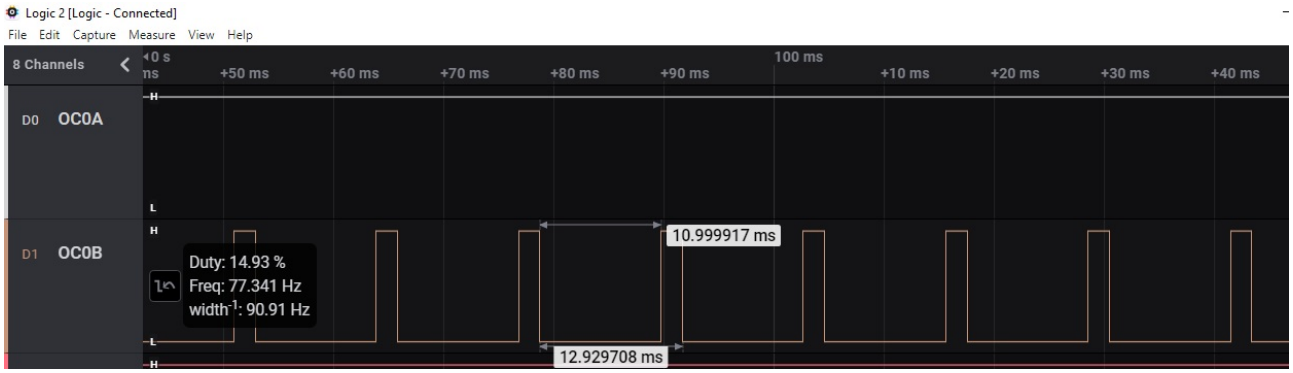


Рисунок 3.7 - Часова діаграма після завантаження **Prog3.4**.

З цієї діаграми бачимо, що на піні OC0A генерується завжди високий сигнал, а на піні OC0B – ШІМ сигнал з частотою 77,341 Гц. Для розрахування тривалості низького сигналу (так как ШІМ інвертований, див. строку 12 **Prog3.4**) скористуємось пропорцією (3.8), де T = 1/77,341 = 12,9297 мс, а TOP = 200. Тоді отримуємо

$$t_1 = \frac{(1+170) \cdot 12{,}9297}{201} = \frac{(1+170) \cdot 12{,}9297}{201} = 10{,}9999 мс . \qquad (3.13)$$

Звернемо увагу на те, що для розрахунку періоду та часу тривалості низького сигналу було взято частоту отриману з експериментів і тоді розраховані значення T та t_1 повністю співпадають з діаграмою на рисунку 3.7.

Розглянемо зараз режими **PWM, Phase Correct**. Це ШІМ режими з фазовою корекцією, вони налаштовуються за допомогою таблиці 3.2, та мають номери 1 та 5. В цих режимах рахунковий регістр TCNT0 спочатку рахує від 0 до TOP, а потім від TOP до нуля. В режимі №1 TOP = 255, а в режимі №5 TOP = OCR0A, тобто максимальне значення до якого рахує таймер знаходиться в регістри OCR0A. Частота ШІМ сигналу в режимах PWM, Phase Correct розраховується за формулою

$$f_{OC0x} = \frac{f_{clk}}{2 \cdot N \cdot TOP}. \quad (3.14)$$

Порівняйте з формулою (3.5). Поведінка пінів OC0A та OC0B залежить від виставлених бітів (Compare Output Mode) COM0A0, COM0A1 та COM0B0, COM0B1 регістра TCCR0A (рисунок 3.1) та визначається згідно з таблиці 3.5 або дивись таблиці технічного опису для мікроконтролеру ATmega328 №19-5 та №19-8, а також для ATmega2560 №75 та №78.

Таблиця 3.5 - Поведінка пінів OC0A, OC0B в режимі PWM, Phase Correct.

COM0*x*1	COM0*x*0	Опис
0	0	Пін OC0*x* відключено від таймера.
0	1	*Для піна OC0A*: якщо WGM02 = 0, пін OC0A відключено від таймера; якщо WGM02 = 1, стан піна OC0A змінюється на протилежний коли значення регістрів TCNT0 і OCR0A співпадають. Для піна OC0B: зарезервовано.
1	0	*Режим неінвертованого ШІМ-сигналу з точною фазою* У разі рівності значень регістрів TCNT0 і OCR0*x* пін OC0*x* скидається в «0» при прямому рахунку і встановлюється в «1» при зворотному рахунку.
1	1	*Режим інвертованого ШІМ-сигналу з точною фазою* У разі рівності значень регістрів TCNT0 і OCR0*x* пін OC0*x* встановлюється в «1» при прямому рахунку і скидається в «0» при зворотному рахунку.

Примітка: *x* = A або B.

Напишемо програму **Prog3.5** в якій налаштовано ШІМ режим з фазовою корекцією № 1 згідно з таблиці 3.2.

Prog3.5

```
#define F_CPU 16000000UL  //  частота - 16MHz
#include <avr/io.h>       // Бібліотека вводу/виводу

int main(void){
  DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
  DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
  TCCR0A |= (1<<WGM00);  //режим PWM Phase Correct №1

  TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
  TCCR0A |= (1<<COM0A1);              //неінвертований ШІМ
  OCR0A = 200; // максимум 255
  TCCR0A |= (1<<COM0B1)|(1<<COM0B0); //інвертований ШІМ
  OCR0B = 170; // максимум 255

    while (1){};        //основна програма
}
```

Розрахуємо частоту ШІМ сигналу по формулі (3.14)

$$f_{OC0x} = \frac{16 \cdot 10^6}{2 \cdot 1024 \cdot 255} = \frac{16 \cdot 10^6}{522240} = 30{,}64 Гц\,. \tag{3.15}$$

Після завантаження маємо наступну часову діаграму (рисунок 3.8). Порівняйте її з діаграмою (рисунок 3.5). Для розрахунку тривалості високого рівня при неінвертованому ШІМ сигналі та тривалості низького рівня при інвертованому складемо пропорцію (порівняйте її з 3.8)

$$\begin{matrix} TOP \leftrightarrow T \\ PWM \leftrightarrow t \end{matrix}\,. \tag{3.16}$$

Рисунок 3.8 - Часова діаграма після завантаження **Prog3.5**.

Щоб отримати як можна більш точніші результати розрахуємо період по експериментальній частоті з рисунку 3.8. Тоді маємо T = 1/30,483 = 32,805 мс. Звідси знаходимо час тривалості високого рівня

$$t_1 = \frac{200 \cdot 32{,}805}{255} = 25{,}729 мс\,, \tag{3.17}$$

та час тривалості низького рівня сигналу

$$t_2 = \frac{170 \cdot 32{,}805}{255} = 21{,}87 мс\,, \tag{3.18}$$

що співпадає зі значенням на рисунку 3.8.

Дослідимо режим фазової корекції №5. Напишемо наступну програму.

Prog3.6

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRD |= (1<<DDD6); //PD6=OC0A=D6 як ВИХІД
6      DDRD |= (1<<DDD5); //PD5=OC0B=D5 як ВИХІД
7      TCCR0A |= (1<<WGM00);  //режим PWM Phase Correct №5
8      TCCR0B |= (1<<WGM02);
9
10     TCCR0B |= (1<<CS02)|(1<<CS00); //дільник 1024
11     TCCR0A |= (1<<COM0A1);              //неінвертований ШІМ
12     OCR0A = 200; // максимум 255
13     TCCR0A |= (1<<COM0B1)|(1<<COM0B0); //інвертований ШІМ
14     OCR0B = 170; // максимум OCR0A
15
16       while (1){};        //основна програма
17   }
```

Згідно з таблицею 3.2 в режимі №5 TOP лічильника визначається регістром OCR0A, а ШІМ параметр для піна OC0B - регістром OCR0B. Таким чином частоту ШІМ сигналу знаходимо по формулі (3.14)

$$f_{OC0x} = \frac{16 \cdot 10^6}{2 \cdot 1024 \cdot 200} = \frac{16 \cdot 10^6}{409600} = 39{,}06 Гц . \qquad (3.19)$$

Завантажимо програму **Prog3.6** до мікроконтролеру ATmega328 та отримуємо часову діаграму зображену на рисунку 3.9.

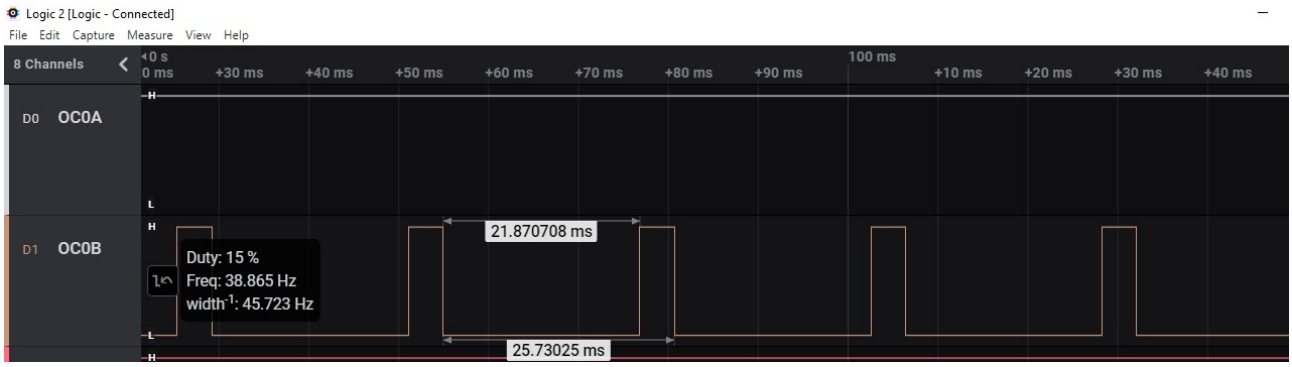


Рисунок 3.9 - Часова діаграма після завантаження **Prog3.6**.

Згідно з пропорцією (3.16) отримуємо тривалість низького рівня ШІМ сигналу

$$t_2 = \frac{170 \cdot 25{,}73}{200} = 21{,}87 мс . \qquad (3.20)$$

Також як і в програмі **Prog3.4** строки 5, 11 та 12 які пов'язані з піном OC0A в програмі **Prog3.6** не важливі, тому що в даному режимі цей пін не використовується.

Звернемо увагу на один важливий момент. Якщо в програмі **Prog3.6** покласти `OCR0B = 0`, то ми отримуємо на піні OC0B сигнал високого рівня, а якщо

`OCR0B = 200`, то – високого рівня. Для того, щоб впевнитись в коректності пропорції (3.16) в ШІМ режимах з фазовою корекцією покладемо в **Prog3.6** `OCR0B = 1`, тоді ми отримуємо при даних налаштуваннях сигнал низького рівня мінімальної тривалості

$$t_2 = \frac{1 \cdot 25{,}73}{200} = 128{,}6 мкс\,. \qquad (3.21)$$

Що підтверджує часова діаграма на рисунку 3.10.

Рисунок 3.10 - Часова діаграма в якій `OCR0B = 1` для **Prog3.6**.

Хід роботи

Завдання №3.1.

Написати програму мовою С(С++) яка налаштовує відповідний режим роботи (таблиця 3.6) потрібного таймеру, при цьому при кожному збігу регістрів OCR*n*A, OCR*n*B та TCNT*n* стан пінів OC*n*A, OC*n*B повинен інвертуватися. Зробіть наступне:

1) розрахуйте теоретично частоту отриманих сигналів;
2) виміряйте експериментально частоту отриманих сигналів;
3) виміряйте час *t* зсуву сигналів (рисунок 3.3 або рисунок 3.4);
4) розрахуйте теоретично час *t* зсуву сигналів (рисунок 3.3 або рисунок 3.4);
5) порівняйте теоретичні та експериментальні данні.

Таблиця 3.6. – Вхідні данні для завдання 3.1

Вариант	Режим, таймер№*n*	Дільник	OCR*n*B, OCR*n*A
1	0, TC0	1	30, 250
2	2, TC0	8	35, 245
3	0, TC2	1	40, 240
4	2, TC2	8	45, 235
5	0, TC0	64	50, 230
6	2, TC0	256	55, 225
7	0, TC2	32	60, 220
8	2, TC2	64	65, 215
9	0, TC0	256	70, 210
10	2, TC0	8	75, 205
11	0, TC2	128	80, 200
12	2, TC2	256	85, 195
13	0, TC0	64	90, 190
14	2, TC0	8	95, 185
15	0, TC2	1024	100, 180
16	2, TC2	32	105, 175
17	0, TC0	8	110, 170
18	2, TC0	256	115, 165
19	0, TC2	128	120, 160
20	2, TC2	64	125, 155

Завдання №3.2.

Написати програму мовою С(С++) яка генерує на піні ОС*n*А інвертований ШІМ сигнал, а на піні ОС*n*В неінвертований ШІМ сигнал відповідної частоти (таблиця 3.7). Розрахувати теоретично та перевірити експериментально ШІМ параметри завдяки яких на піні ОС*n*А генерується сигнал високого рівня заданої тривалості (таблиця 3.7), а на піні ОС*n*В - сигнал низького рівня заданої тривалості (таблиця 3.7). В звіті представити всі необхідні лістинги програм, формули та розрахунки, а також скріншоти програми логічного аналізатору.

Таблиця 3.7. – Вхідні данні для завдання 3.2

Вариант	Режим, таймер№*n*	Частота, Гц	t_{OCnA}, t_{OCnB}, мкс
1	1, TC2	980,39	400, 270
2	3, TC2	976,56	300, 1000
3	1, TC0	490,196	1850, 1400
4	3, TC0	244,14	3000, 1100
5	1, TC2	245,098	770, 1200
6	3, TC2	244,14	800, 1500
7	1, TC0	3921,57	100, 210
8	3, TC0	62500	3, 10
9	1, TC2	30,637	5870, 2200
10	3, TC2	62500	5, 8
11	1, TC0	122,55	4900, 550
12	3, TC0	976,56	780, 80
13	1, TC2	3921,57	190, 50
14	3, TC2	1953,1	420, 150
15	1, TC0	490,196	720, 2000
16	3, TC0	7812,5	110, 35
17	1, TC2	245,098	3900, 2100
18	3, TC2	244,14	1200, 3500
19	1, TC0	30,637	7510, 1050
20	3, TC0	244,14	4000, 2700

Завдання №3.3.

Написати програму мовою С(С++) яка генерує на піні ОС*n*В ШІМ сигнал відповідної частоти (таблиця 3.8). Тривалість високого (для інвертованого ШІМ) та низького (для неінвертованого ШІМ) визначається в процентах від тривалості низького сигналу для інвертованого ШІМ та від тривалості високого сигналу для неінвертованого ШІМ.

Таблиця 3.8. – Вхідні данні для завдання 3.3

Вариант	Режим, таймер№*n*, ШІМ	Частота, Гц	Дільник	%
1	5, TC2, I	400000	1	25
2	7, TC0, HI	2840,91	256	10
3	7, TC2, I	43478,3	8	15
4	5, TC0, HI	1041,67	256	20
5	5, TC2, I	7142,86	32	40
6	7, TC0, HI	16129	8	55
7	7, TC2, I	3968,25	64	5
8	5, TC0, HI	25000	8	60
9	5, TC2, I	1388,89	128	50
10	7, TC0, HI	1262,63	64	65
11	7, TC2, I	183,824	1024	70
12	5, TC0, HI	1785,71	64	25
13	5, TC2, I	260,417	256	60
14	7, TC0, HI	91428,6	1	75
15	7, TC2, I	2525,25	32	80
16	5, TC0, HI	61538,5	1	30
17	5, TC2, I	4629,63	64	35
18	7, TC0, HI	18018	8	85
19	7, TC2, I	939,85	128	90
20	5, TC0, HI	17241,4	8	45

Примітка: I – інвертований ШІМ, HI – неінвертований ШІМ.

Під час виконання кожного наступного завдання рекомендується змінити мікроконтролер. Так, якщо студент закінчив виконувати другу лабораторну роботу з ATmega328, то завдання 3.1 він має виконати на ATmega2560, завдання 3.2 на ATmega328 та завдання 3.3 на ATmega2560, та навпаки.

Лабораторна робота №4

16-бітні таймер-лічильники

Мета: навчитися налаштовувати 16-бітні таймер-лічильники мікроконтролерів ATmega328 і ATmega2560 в різні режими за допомогою регістрів, а також теоретично розраховувати та експериментально визначати параметри роботи.

Теоретичні відомості

16-бітні таймер-лічильники дуже схожі на 8-бітні але відрізняються від них більшими можливостями. В мікроконтролері ATmega328 є тільки один 16-бітний таймер-лічильник це Timer/Counter1, а в мікроконтролері ATmega2560 таких таймер-лічильників чотири це Timer/Counter1, Timer/Counter3, Timer/Counter4 та Timer/Counter5. Таймер-лічильник №1 в обох мікроконтролерах відрізняються тільки тим, що в ATmega328 він має два порівняльних регістра (канали A і B), а в ATmega2560 порівняльних регістрів три (канали A, B і C). В цій лабораторній роботі будемо говорити про узагальнений таймер-лічильник №*n*, де *n* =1, 3, 4, 5 якщо мова йде про ATmega2560 та *n* =1 якщо мова йде про ATmega328.

Таким чином в 16-бітних таймерах є 16-бітний рахунковий регістр TCNT*n* (Timer/Counter Register) який має можливість рахувати імпульси від 0 до 65535.

Кожного разу значення рахункового регістра порівнюється зі значеннями які знаходяться в 16-бітних порівняльних регістрах OCR*n*A, OCR*n*B і OCR*n*C (Output Compare Register A, B, C). Ці регістри зв'язані з відповідними пінами OC*nx*, де *x*=A, B, C. Зв'язок між OC*nx* та пінами GPIO вказано в таблиці 4.1.

Таблиця 4.1 - Відповідність пінів OC*nx* до пінів GPIO.

Піни	**ATmega328**		**ATmega2560**	
OC1A	PB1	D9	PB5	D11
OC1B	PB2	D10	PB6	D12
OC1C			PB7	D13
OC3A			PE3	D5
OC3B			PE4	D2
OC3C			PE5	D3
OC4A			PH3	D6
OC4B			PH4	D7
OC4C			PH5	D8
OC5A			PL3	D46
OC5B			PL4	D45
OC5C			PL5	D44

Для налаштування режимів роботи існують 8-бітні регістри TCCR*n*A, TCCR*n*B і TCCR*n*C (Timer/Counter Control Register A, B, C). Структура регістра TCCR1A мікроконтролера ATmega328 (дивись пункт 20.15.1 технічної документації) та регістрів TCCR*n*A мікроконтролера ATmega2560 (дивись сторінку 160 технічної документації) приведена на рисунку 4.1.

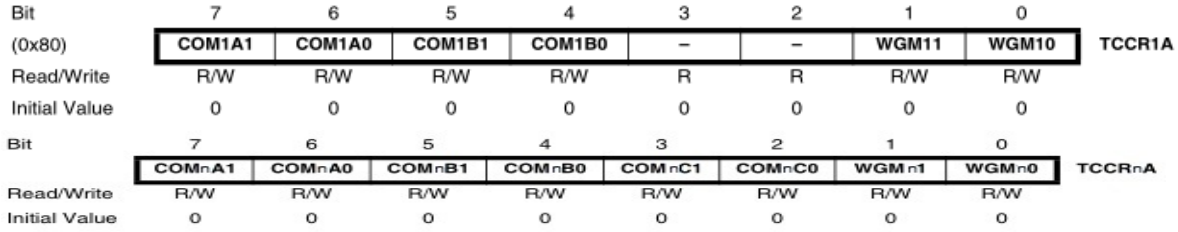


Рисунок 4.1 - Структура регістрів TCCR*n*A.

Біти COM*n*A1 та COM*n*A0 слугують для налаштування піна OC*n*A каналу А таймеру №n, біти COM*n*B1 та COM*n*B0 налаштовують пін OC*n*B каналу В таймеру №n. В мікроконтролері ATmega2560 в таймерах 1, 3, 4, 5 існує додатковий канал С (пін OC*n*C), який налаштовується бітами COM*n*C1 та COM*n*C0.

Регістри TCCR*n*B мікроконтролерів ATmega328 та ATmega2560 не відрізняються і представлені на рисунку 4.2.

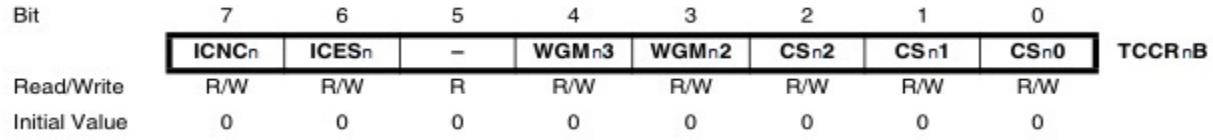


Рисунок 4.2 - Структура регістрів TCCR*n*B.

Біти CS*n*2, CS*n*1 та CS*n*0 налаштовують частоту роботи таймеру №*n* згідно з таблиці 4.2.

Таблиця 4.2 - Налаштування частоти таймер-лічильників.

CS*n*2	CS*n*1	CS*n*0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	clkI/O/1 No prescaling
0	1	0	clkI/O/8 (From prescaler)
0	1	1	clkI/O/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	clkI/O/1024 (From prescaler)
1	1	0	External clock source on T*n* pin. Clock on falling edge
1	1	1	External clock source on T*n* pin. Clock on rising edge

n=1,3,4,5.

Піни T*n* на які можуть подаватися зовнішні імпульси (останні два рядки таблиці 4.2) відповідно до технічної документації:

ATmega328 T1 = PD5 (D5);

ATmega2560 T1 = PD6 (-), T3 = PE6 (-), T4 = PH7 (-), T5 = PL2 (D47).

Тут у круглих дужках вказані піни платформ Arduino UNO та Arduino MEGA, бачимо, що PD6, PE6, PH7 не виведені на роз'єм відповідного модуля.

За допомогою бітів WGM (**Waveform Generation Mode**), а саме WGM*n*0, WGM*n*1 (регістр TCCR*n*A) та WGM*n*2, WGM*n*3 (регістр TCCR*n*B) налаштовуються режими роботи які представлені в таблиці 4.3 (Table 20-6 для ATmega328, Table 82 для ATmega2560).

Таблиця 4.3 - Режими роботи 16-бітних таймер-лічильників

№	WGMn3	WGMn2	WGMn1	WGMn0	Режим	TOP
0	0	0	0	0	Звичайний режим	0xFFFF
1	0	0	0	1	Корекція фази ШІМ, 8-біт	0x00FF
2	0	0	1	0	Корекція фази ШІМ, 9-біт	0x01FF
3	0	0	1	1	Корекція фази ШІМ, 10-біт	0x03FF
4	0	1	0	0	Скидання при збігу, CTC	OCR*n*A
5	0	1	0	1	Швидкий ШІМ, 8-біт	0x00FF
6	0	1	1	0	Швидкий ШІМ, 9-біт	0x01FF
7	0	1	1	1	Швидкий ШІМ, 10-біт	0x03FF
8	1	0	0	0	ШІМ, корекція фази та частоти	ICR*n*
9	1	0	0	1	ШІМ, корекція фази та частоти	OCR*n*A
10	1	0	1	0	ШІМ, корекція фази	ICR*n*
11	1	0	1	1	ШІМ, корекція фази	OCR*n*A
12	1	1	0	0	Скидання при збігу, CTC	ICR*n*
13	1	1	0	1	Не використовується	–
14	1	1	1	0	Швидкий ШІМ	ICR*n*
15	1	1	1	1	Швидкий ШІМ	OCR*n*A

З цієї таблиці бачимо, що у 16-бітних таймерів режимів значно більше в порівнянні з режимами 8-бітних таймерів, але можна виділити чотири типи: звичайний, CTC, швидкий ШІМ, ШІМ з корекцією фази. У звичайному режимі №0 лічильник рахує від 0 до 65535, а потім значення рахункового регістра TCNT*n* скидається в нуль.

Режимів CTC (**Clear Timer on Compare Match**) – скидання при збігу зараз два. В режимі №4 скидання регістру TCNT*n* відбувається при збігу значень

TCNT*n* та OCR*n*A, а режимі №12 скидання регістру TCNT*n* відбувається при збігу значень TCNT*n* та ICR*n*. Регістр ICR*n* (**Input Capture Register**) це додатковий 16-бітний регістр захоплення.

Режими швидкого ШІМ. Це режими №5, 6, 7, 14, 15. В режимах №5, 6, 7 верхня межа (TOP), яка визначає частоту ШІМ сигналу та кількість градацій, фіксована і дорівнює відповідно 11111111_2 = 255, 111111111_2 = 511, 1111111111_2 = 1023. В режимі №14 верхня межа визначається регістром ICR*n*, а в режимі №15 – регістром OCR*n*A. Після досягнення рахунковим регістром TCNT*n* значення TOP його значення скидається в нуль.

В ШІМ режимах з корекцією фази №1, 2, 3, 8, 9, 10, 11 значення рахункового регістру TCNTn спочатку збільшуються від нуля до TOP, а потім зменшуються від TOP до нуля. В ШІМ режимі з корекцією фази та частоти регістр порівняння оновлюється, коли значення рахункового регістру досягає мінімуму. В ШІМ режимі з корекцією фази регістр порівняння оновлюється, коли значення рахункового регістру досягає максимуму.

Розглянемо можливі налаштування пінів OC*n*A/OC*n*B/OC*n*C (у випадку ATmega328 відсутній OC*n*C) в різних режимах.

Таблиця 4.4 - Поведінка виводів OC*n*A/OC*n*B/OC*n*C у режимах без ШІМ.

COM*n*A1 COM*n*B1 COM*n*C1	COM*n*A0 COM*n*B0 COM*n*C0	Опис
0	0	Виводи OC*n*A/OC*n*B/OC*n*C відключені від таймера.
0	1	Стан OC*n*A/OC*n*B/OC*n*C виводів змінюється на протилежний
1	0	Стан OC*n*A/OC*n*B/OC*n*C скидається в «0»
1	1	Стан OC*n*A/OC*n*B/OC*n*C встановлюються в «1»

Таблиця 4.5 - Поведінка виводів OCnA/OCnB/OCnC у режимах Fast PWM.

COM*n*A1 COM*n*B1 COM*n*C1	COM*n*A0 COM*n*B0 COM*n*C0	Опис
0	0	Виводи OC*n*A/OC*n*B/OC*n*C відключені від таймера.
0	1	Якщо біти WGM*n*3 – WGM*n*0 встановлені (0000 – 1101), виводи OCnA/OCnB/OCnC не функціонують. Якщо біти WGM*n*3 – WGM*n*0 встановлені в 1110 або 1111, стан виводу OC*n*A змінюється на протилежний при збігу з OCR*n*A. Піни OC*n*B та OC*n*C відключені від таймера.
1	0	Режим неінвертованого ШІМ-сигналу. OC*nx* скидається в «0» якщо TCNT*n* = OCR*nx*. Для ATmega328: OC*nx* встановлюється в «1» якщо TCNT*n* = 0x00. Для ATmega2560: OC*nx* встановлюється в «1» якщо TCNT*n* = 0xFF.
1	1	Режим інвертованого ШІМ-сигналу. OC*nx* встановлюється в «1» якщо TCNT*n* = OCR*nx*. Для ATmega328p: OC*nx* скидається в «0» якщо TCNT*n* = 0x00. Для ATmega2560: OC*nx* скидається в «0» за рівності TCNT*n* = 0xFF

Примітка: *x* = A, B або C.

Таблиця 4.6 - Поведінка виводів OC*n*A/OC*n*B/OC*n*C у режимах PWM з фазовою або фазово-частотною корекцією.

COM*n*A1 COM*n*B1 COM*n*C1	COM*n*A0 COM*n*B0 COM*n*C0	Опис
0	0	Виводи OC*nx* відключені від таймера.
0	1	Для ATmega328 Якщо біти WGM13 – WGM10 встановлені в 1001 або 1011, стан виводу OC1A змінюється на протилежний при збігу з OCR1A. Пін OC1B відключено від таймера. Для ATmega2560 Якщо біти WGM*n*3 – WGM*n*0 встановлені від 1000 до 1011, стан виводів OC*n*A змінюється на протилежний при збігу з OCR*n*A. Піни OC*n*B, OC*n*C відключені від таймера.

COM*n*A1 COM*n*B1 COM*n*C1	COM*n*A0 COM*n*B0 COM*n*C0	Опис
		Для ATmega328p та ATmega2560 В інших випадках виводи OC*n*A/OC*n*B/OC*n*C відключені від таймера
1	0	Режим неінвертованого ШИМ-сигналу OC*nx* скидається в «0» якщо TCNT*n* = OCR*nx* під час збільшення значення лічильника. OC*nx* встановлюється в «1» якщо TCNT*n* = OCR*nx* під час зменшення значення лічильника.
1	1	Режим інвертованого ШИМ-сигналу OC*nx* встановлюється в «1» якщо TCNT*n* = OCR*nx* під час збільшення значення лічильника. OC*nx* скидається в «0» якщо TCNT*n* = OCR*nx* під час зменшення значення лічильника.

Примітка: *x* = A, B або C.

Приведемо приклад програми яка встановлює звичайний режим (режим №0) роботи таймер-лічильника 5 на мікроконтролері ATmega2560.

Prog4.1

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRL |= (1<<DDL3); //PL3=OC5A=D46 як ВИХІД
6      DDRL |= (1<<DDL4); //PL4=OC5B=D45 як ВИХІД
7      DDRL |= (1<<DDL5); //PL5=OC5C=D44 як ВИХІД
8      TCCR5B |= (1<<CS50);   //дільник N=1
9      TCCR5A |= (1<<COM5A0); //переключення OC5A при збігу OCR5A=TCNT5
10     TCCR5A |= (1<<COM5B0); //переключення OC5B при збігу OCR5B=TCNT5
11     TCCR5A |= (1<<COM5C0); //переключення OC5C при збігу OCR5C=TCNT5
12     OCR5A = 10000; // максимум 65535
13     OCR5B = 20000; // максимум 65535
14     OCR5C = 35000; // максимум 65535
15       while (1){};        //основна програма
16   }
```

Завантаживши **Prog4.1** до мікроконтролеру отримуємо часову діаграму зображену на рисунку 4.3. Розрахуємо теоретично характеристики отриманих сигналів. Згідно з формулою (3.1) час одного такту таймеру дорівнює

$$t_{такт} = N \cdot 62{,}5нс\,, \tag{4.1}$$

де N – дільник згідно таблиці 4.2. В звичайному режимі (який встановлено за замовчуванням в **Prog4.1**) таймер рахує від 0 до 65535 тобто робить 65536 тактів, таким чином тривалість високого або низького сигналу буде дорівнювати

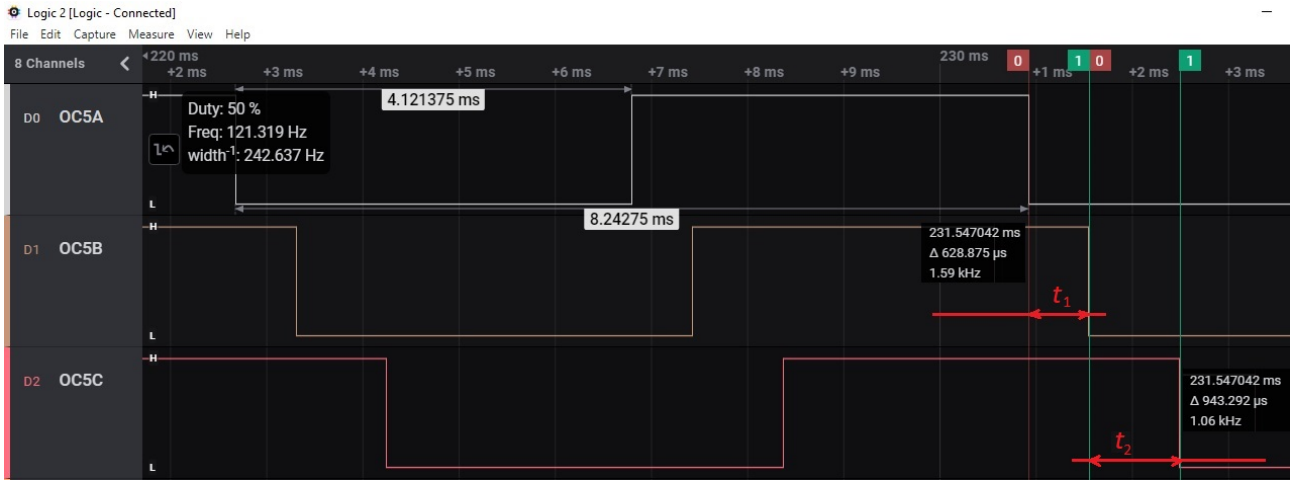


Рисунок 4.3 - Часова діаграма після завантаження **Prog4.1**.

$$t = 65536 \cdot 62{,}5нс = 4{,}096мс\,, \tag{4.2}$$

де враховано, що в нашому випадку N=1. Отримане теоретичне значення доволі близько до експериментального (див. рисунок 4.2). Для розрахунку часового зсуву t_1 між сигналами на пінах OC5A та OC5B треба помножити кількість тактів на час одного такту

$$t_1 = (20000 - 10000) \cdot 62{,}5нс = 625мкс\,. \tag{4.3}$$

Аналогічно

$$t_2 = (35000 - 20000) \cdot 62{,}5нс = 937{,}5мкс\,. \tag{4.4}$$

Розглянемо режими СТС №4 та №12 програми **Prog4.2** та **Prog4.3** відповідно.

Prog4.2

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){
5      DDRL |= (1<<DDL3); //PL3=OC5A=D46 як ВИХІД
6      DDRL |= (1<<DDL4); //PL4=OC5B=D45 як ВИХІД
7      DDRL |= (1<<DDL5); //PL5=OC5C=D44 як ВИХІД
8      TCCR5B |= (1<<WGM52);  //режим СТС №4
9      TCCR5B |= (1<<CS50);   //дільник N=1
10     TCCR5A |= (1<<COM5A0); //переключення OC5A при збігу OCR5A=TCNT5
11     TCCR5A |= (1<<COM5B0); //переключення OC5B при збігу OCR5B=TCNT5
12     TCCR5A |= (1<<COM5C0); //переключення OC5C при збігу OCR5C=TCNT5
13     OCR5A = 30000; // максимум 65535
14     OCR5B = 20000; // максимум OCR5A
15     OCR5C = 5000;  // максимум OCR5A
16       while (1){};        //основна програма
17   }
```

Prog4.3

```
#define F_CPU 16000000UL  //  частота - 16MHz
#include <avr/io.h>       // Бібліотека вводу/виводу

int main(void){
  DDRL |= (1<<DDL3); //PL3=OC5A=D46 як ВИХІД
  DDRL |= (1<<DDL4); //PL4=OC5B=D45 як ВИХІД
  DDRL |= (1<<DDL5); //PL5=OC5C=D44 як ВИХІД
  TCCR5B |= (1<<WGM52)|(1<<WGM53);  //режим СТС №12
  TCCR5B |= (1<<CS50);   //дільник N=1
  TCCR5A |= (1<<COM5A0); //переключення OC5A при збігу OCR5A=TCNT5
  TCCR5A |= (1<<COM5B0); //переключення OC5B при збігу OCR5B=TCNT5
  TCCR5A |= (1<<COM5C0); //переключення OC5C при збігу OCR5C=TCNT5
  ICR5 = 30000;  // максимум 65535
  OCR5A = 30000; // максимум ICR5
  OCR5B = 20000; // максимум ICR5
  OCR5C = 5000;  // максимум ICR5
    while (1){};        //основна програма
}
```

Частота сигналу розраховується за формулою

$$f_{OCnx} = \frac{f_{clk}}{2 \cdot N \cdot (TOP + 1)}, \tag{4.5}$$

де TOP=OCR*n*A у випадку режиму №4 та TOP=ICR*n* у випадку режиму №12 якщо налаштовано інвертування стану піна при збігу значень рахункового та порівняльного регістрів. Розрахунок по (4.5) для програм **Prog4.2** або **Prog4.3** призводить до

$$f_{OCnx} = \frac{16 \cdot 10^6}{2 \cdot 1 \cdot (30000 + 1)} = 266{,}66 Гц, \tag{4.6}$$

що близько до експериментального значення (див. рисунок 4.4).

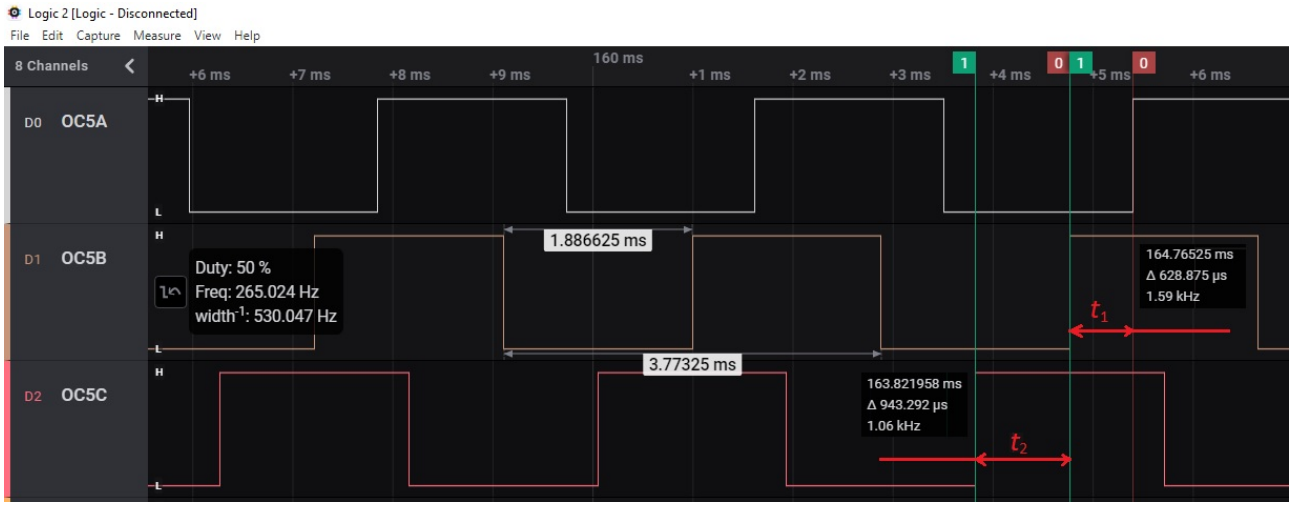


Рисунок 4.4 - Часова діаграма після завантаження **Prog4.2** або **Prog4.3**.

Часовий зсув t_1 та t_2 розраховується по формулам подібним (4.3) та (4.4)

$$t_1 = (30000 - 20000) \cdot 62{,}5нс = 625мкс, \tag{4.7}$$

$$t_2 = (20000 - 5000) \cdot 62{,}5нс = 937{,}5мкс. \tag{4.8}$$

Розглянемо режими Fast PWM, тобто швидкого ШІМ (широтна імпульсна модуляція). Згідно таблиці 4.3 у 16-бітних таймерів таких режимів п'ять: №5, 6, 7, 14, 15. В режимі швидкого ШІМ лічильник рахує від нуля до TOP. Після цього значення рахункового регістра скидається в нуль. Значення TOP залежить від обраного режиму (дивись таблицю 4.3). Частота ШІМ сигналу в режимах Fast PWM розраховується за формулою

$$f_{OCnx} = \frac{f_{clk}}{N \cdot (1 + TOP)}. \tag{4.9}$$

Поведінка пінів OC1A та OC1B мікроконтролера ATmega328, а також пінів OC*n*A, OC*n*B та OC*n*C мікроконтролера ATmega2560 залежить від виставлених бітів (**Compare Output Mode**) COM*n*A0, COM*n*A1, COM*n*B0, COM*n*B1 COM*n*C0, COM*n*C1 регістра TCCR*n*A (див. рисунок 4.1) та визначається згідно з таблицею 4.5. Тривалість високого рівня сигналу *t* у разі неінвертованого ШІМ розраховується за пропорцією

$$\begin{matrix} 1 + TOP \leftrightarrow T \\ 1 + PWM \leftrightarrow t \end{matrix}, \tag{4.10}$$

де *PWM* – ШІМ параметр (тобто значення регістрів OCR*nx*), $T = 1/f_{OCnx}$ – період ШІМ сигналу. Якщо використовується інвертований ШІМ, то пропорція (4.10) визначає тривалість *t* низького рівня сигналу.

Приведемо приклад роботи в режимі Fast PWM №5 (дивись таблицю 4.3) на п'ятому таймері.

Prog4.4

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){      //ATmega2560
5      DDRL |= (1<<DDL3); //PL3=OC5A=D46 як ВИХІД
6      DDRL |= (1<<DDL4); //PL4=OC5B=D45 як ВИХІД
7      DDRL |= (1<<DDL5); //PL5=OC5C=D44 як ВИХІД
8      TCCR5A |= (1<<WGM50);      //режим FAST PWM №5
9      TCCR5B |= (1<<WGM52);
10
11     TCCR5B |= (1<<CS52)|(1<<CS50);   //дільник N=1024
12     TCCR5A |= (1<<COM5A1);             //неінвертований ШІМ OC5A
13     TCCR5A |= (1<<COM5B1)|(1<<COM5B0); //інвертований ШІМ OC5B
14     TCCR5A |= (1<<COM5C1);             //неінвертований ШІМ OC5C
15     OCR5A = 100; // максимум 255
16     OCR5B = 0;   // максимум 255
17     OCR5C = 0;   // максимум 255
18       while (1){};        //основна програма
19   }
```

Після завантаження отримуємо наступну часову діаграму (див. рисунок 4.5).

Рисунок 4.5 - Часова діаграма після завантаження **Prog4.4**.

По-перше відмітимо, що на каналах A, C налаштовано неінвертований ШІМ, а на каналі B – інвертований. Згідно з формулою (4.9) частота ШІМ сигналу дорівнює

$$f_{OCnx} = \frac{16 \cdot 10^6}{1024 \cdot (1+255)} = 61{,}04 Гц\,, \tag{4.11}$$

що відповідає періоду $T = 1/61{,}04 = 16{,}4 мс$. Таким чином з (4.10) знаходимо для тривалості високого рівня сигналу на каналі A

$$t_A = \frac{(1+100) \cdot 16{,}4}{1+255} = 6{,}46 мс\,, \tag{4.12}$$

тривалість низького рівня сигналу на каналі B

$$t_B = \frac{(1+0) \cdot 16{,}4}{1+255} = 64 мкс\,, \tag{4.13}$$

тривалість високого рівня сигналу на каналі C

$$t_C = \frac{(1+0) \cdot 16{,}4}{1+255} = 64 мкс\,. \tag{4.14}$$

Нажаль, тривалості t_B, t_C на рисунку 4.5 ми не можемо побачити, але їх можна розрахувати з цього рисунка

$$t_B = t_C = 16{,}484958 мс - 16{,}420583 мс = 0{,}064375 мс = 64{,}375 мкс\,, \tag{4.15}$$

що достатньо близько до (4.13), (4.14).

Розглянемо режими ШІМ з фазовою корекцією. Згідно таблиці 4.3 у 16-бітних таймерів таких режимів сім: №1, 2, 3, 8, 9, 10, 11. В ШІМ режимі з фазовою

корекцією лічильник рахує спочатку від нуля до TOP, а потім від TOP до нуля. Для режимів №1, 2, 3 значення TOP фіксовано і дорівнює відповідно $11111111_2 = 255$, $111111111_2 = 511$, $1111111111_2 = 1023$. В режимах №8, 10 значення TOP визначається регістром ICR*n*, а в режимах №9, 11 - регістром OCR*n*A. Частота сигналу в ШІМ режимах з фазовою корекцією розраховується за формулою

$$f_{OCnx} = \frac{f_{clk}}{2 \cdot N \cdot TOP}. \tag{4.16}$$

Поведінка пінів OC1A та OC1B мікроконтролера ATmega328, а також пінів OC*n*A, OC*n*B та OC*n*C мікроконтролера ATmega2560 залежить від виставлених бітів (**Compare Output Mode**) COM*n*A0, COM*n*A1, COM*n*B0, COM*n*B1 COM*n*C0, COM*n*C1 регістра TCCR*n*A (див. рисунок 4.1) та визначається згідно з таблиці 4.6. Тривалість високого рівня сигналу *t* у разі неінвертованого ШІМ розраховується за пропорцією

$$\begin{matrix} TOP \leftrightarrow T \\ PWM \leftrightarrow t \end{matrix}, \tag{4.17}$$

де *PWM* – ШІМ параметр (тобто значення регістрів OCR*nx*), $T = 1/f_{OCnx}$ – період ШІМ сигналу. Якщо використовується інвертований ШІМ, то пропорція (4.16+1) визначає тривалість *t* низького рівня сигналу.

Приведемо приклад роботи в ШІМ режимі з фазовою корекцією №1 (дивись таблицю 4.3) на п'ятому таймері.

Prog4.5

```
1    #define F_CPU 16000000UL  //  частота - 16MHz
2    #include <avr/io.h>       // Бібліотека вводу/виводу
3
4    int main(void){      //ATmega2560
5      DDRL |= (1<<DDL3); //PL3=OC5A=D46 як ВИХІД
6      DDRL |= (1<<DDL4); //PL4=OC5B=D45 як ВИХІД
7      DDRL |= (1<<DDL5); //PL5=OC5C=D44 як ВИХІД
8      TCCR5A |= (1<<WGM50);      //режим PWM з фазовою корекцією №1
9
10     TCCR5B |= (1<<CS52)|(1<<CS50);   //дільник N=1024
11     TCCR5A |= (1<<COM5A1);             //неінвертований ШІМ OC5A
12     TCCR5A |= (1<<COM5B1)|(1<<COM5B0); //інвертований ШІМ OC5B
13     TCCR5A |= (1<<COM5C1);             //неінвертований ШІМ OC5C
14     OCR5A = 100; // максимум 255
15     OCR5B = 1;   // максимум 255
16     OCR5C = 1;   // максимум 255
17       while (1){};        //основна програма
18   }
```

Після завантаження отримуємо наступну часову діаграму

Рисунок 4.6 - Часова діаграма після завантаження **Prog4.5**.

По-перше відмітимо, що на каналах A, C налаштовано неінвертований ШІМ, а на каналі B – інвертований. Згідно з формулою (4.16) частота ШІМ сигналу дорівнює

$$f_{OCnx} = \frac{16 \cdot 10^6}{2 \cdot 1024 \cdot 255} = 30{,}637 Гц, \tag{4.18}$$

що відповідає періоду $T = 1/30{,}637 = 32{,}64 мс$. Таким чином з (4.17) знаходимо для тривалості високого рівня сигналу на каналі A

$$t_A = \frac{100 \cdot 32{,}64}{255} = 12{,}8 мс, \tag{4.19}$$

тривалість низького рівня сигналу на каналі B

$$t_B = \frac{1 \cdot 32{,}64}{255} = 128 мкс, \tag{4.20}$$

тривалість високого рівня сигналу на каналі C

$$t_C = \frac{1 \cdot 32{,}64}{255} = 128 мкс. \tag{4.21}$$

Тривалості t_B, t_C з рисунку 4.6 можна розрахувати

$$t_B = t_C = 32{,}840583 мс - 32{,}711792 мс = 0{,}128791 мс = 128{,}791 мкс, \tag{4.22}$$

що достатньо близько до (4.20), (4.21).

Хід роботи

Завдання №4.1.

Написати програму мовою С(С++) яка налаштовує відповідний режим роботи (таблиця 4.7) першого таймеру, при цьому при кожному збігу регістрів OCR1A, OCR1B та TCNT1 стан пінів OC1A, OC1B повинен інвертуватися. Зробіть наступне:

1) розрахуйте теоретично частоту отриманих сигналів;
2) виміряйте експериментально частоту отриманих сигналів;
3) виміряйте час *t* зсуву сигналів на пінах OC1A, OC1B;
4) розрахуйте теоретично час *t* зсуву сигналів;
5) порівняйте теоретичні та експериментальні данні.

Таблиця 4.7. – Вхідні данні для завдання 4.1

Вариант	Режим, дільник	ICR1	OCR1A, OCR1B
1	12, 1024	700	500, 200
2	0, 64		11000, 7300
3	4, 8		1800, 1100
4	12, 256	1500	1200, 750
5	0, 1		18500, 12300
6	4, 1024		550, 340
7	12, 64	12000	8500, 3000
8	0, 8		22000, 16000
9	4, 256		900, 650
10	12, 1	55000	42000, 28000
11	0, 1024		31000, 26000
12	4, 64		9500, 6700
13	12, 8	32000	24000, 15000
14	0, 256		37000, 29000
15	4, 1		47000, 33000
16	12, 256	800	650, 380
17	0, 1024		51000, 42000
18	4, 8		29000, 25000
19	12, 64	9000	7500, 5300
20	4, 256		1300, 770

Завдання №4.2.

Налаштувати на пінах OC1A, OC1B ШІМ сигнал у відповідному режимі з вказаною частотою згідно свого варіанта (дивись таблицю 4.8). На піні OC1A встановити інвертований ШІМ з тривалістю високого сигналу як найближче до вказаного в таблиці. На піні OC1B встановити неінвертований ШІМ з тривалістю низького сигналу як найближче до вказаного в таблиці. Перевірити роботу мікроконтролеру за допомогою логічного аналізатору. Розрахувати відносні та абсолютні похибки між теоретичними та експериментальними результатами.

Таблиця 4.8. – Вхідні данні для завдання 4.2

Варіант	Режим №	Частота ШІМ, Гц	t_{pulse} на OC1A, мс	t_{pulse} на OC1B, мс
1	2	1956,95	0,137	0,45
2	3	122,19	2,341	7,853
3	6	3906,25	0,023	0,157
4	7	15625	0,00345	0,0558
5	2	15655,6	0,0438	0,0234
6	3	30,547	6,892	28,428
7	6	30,5176	3,867	29,56
8	7	1953,13	0,274	0,492
9	2	244,618	0,923	2,789
10	3	7820,14	0,0198	0,119
11	6	488,281	0,0545	1,694
12	7	244,141	0,673	2,159
13	2	61,1546	3,562	14,271
14	3	7,637	22,333	101,55
15	6	122,07	1,879	7,937
16	7	61,0352	2,361	14,337
17	2	15,2886	15,348	55,127
18	3	977,517	0,0859	0,928
19	6	31250	0,00948	0,0299
20	7	15,2588	12,94	47,38

Завдання №4.3.

Згенерувати інверсний ШІМ сигнал завданою розрядністю на піні OC1B у відповідному режимі роботи (див. таблицю 4.9). Визначити частоту отриманого сигналу теоретично та експериментально при вказаному дільнику. Розрахувати значення регістра OCR1B таким щоб коефіцієнт заповнення (duty) відповідав процентам в таблиці 4.9. Перевірити розрахунки за допомогою логічного аналізатору.

Таблиця 4.9. – Вхідні данні для завдання 4.3

Варіант	Режим №	Розрядність ШІМ сигналу	Дільник	Коефіцієнт заповнення, %
1	8	15	1	20
2	9	12	8	25
3	10	11	64	27
4	11	14	256	12
5	11	12	1024	32
6	10	14	8	37
7	9	13	1	35
8	8	14	8	30
9	9	15	64	45
10	10	15	1	47
11	11	11	8	42
12	8	13	64	40
13	10	12	256	77
14	11	13	1	72
15	8	12	256	60
16	9	14	1024	55
17	11	15	64	82
18	8	11	1024	70
19	9	11	256	65
20	10	13	1024	87

Лабораторна робота №5

Переривання від таймер-лічильників

Мета: навчитися теоретично розраховувати та експериментально визначати параметри роботи переривань таймер-лічильників мікроконтролерів ATmega328 і ATmega2560 в різних режимах роботи.

Теоретичні відомості

Згідно з технічною документацією у мікроконтролерах реалізовані переривання від таймер-лічильників за такими подіями:

- збіг таймер-лічильника з регістром порівняння;
- переповнення таймер-лічильника;
- захоплення таймер-лічильника.

Захоплення таймер-лічильника є тільки в 16-бітних таймерах. Кількість переривань по переповненню дорівнює кількості таймер-лічильників у мікроконтролері. Кількість переривань за збігом дорівнює повній кількості каналів, в ATmega328 їх 6, в ATmega2560 - 16 (по 3 канали в 4-х шістнадцяти бітних таймерах, по 2 канали в 2-х восьми бітних таймерах).

Для дозволу цих переривань використовуються регістри (**Timer Interrupt Mask Register**) TIMSK*n*, де *n*=0, 1, 2 для ATmega328 та *n*=0, 1, 2, 3, 4, 5 для ATmega2560. Структура регістрів TIMSK*n* для 8-бітних таймерів зображена на рисунку 5.1.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE*n*B	OCIE*n*A	TOIE*n*	TIMSK*n*
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 5.1 - Регістри TIMSK*n*, де *n* = 0 або 2.

Структура регістрів TIMSK*n* для 16-бітних таймерів мікроконтролера ATmega2560 зображена на рисунку 5.2.

Bit	7	6	5	4	3	2	1	0	
	–	–	ICIE*n*	–	OCIE*n*C	OCIE*n*B	OCIE*n*A	TOIE*n*	TIMSK*n*
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 5.2 - Регістри TIMSK*n* мікроконтролера ATmega2560 (*n*=1,3,4,5).

Регістри TIMSK*n* для 8-бітних таймерів мікроконтролерів ATmega328 та ATmega2560 мають однакову структуру. Структура регістрів TIMSK*n* для 16-бітних таймерів мікроконтролерів ATmega328 та ATmega2560 відрізняється тільки тім, що таймери мікроконтролера ATmega328 не мають каналу C.

Біти OCIE*nx* дозволяють переривання при збігу лічильного регістра з відповідним регістром порівняння, біти TOIE*n* дозволяють переривання при переповненні рахункового регістра, а біти ICIE*n* дозволяють переривання по захопленню.

Всі джерела переривань можна подивитися в офіційній технічній документації в таблиці Interrupt Vectors in ATmega328 (Table 16-1) для мікроконтролера ATmega328 та в таблиці Interrupt Vectors in ATmega640/1280/1281/2560/2561 (Table 32) для мікроконтролера ATmega2560. Наведемо тут тільки переривання за таймерами (дивись таблиці 5.1 та 5.2).

Таблиця 5.1. – Вектори переривань за таймерами для ATmega328

Номер вектору	Адреса	Джерело	Опис переривання
8	0x000E	TIMER2 COMPA	Збіг A таймера/лічильника T2
9	0x0010	TIMER2 COMPB	Збіг B таймера/лічильника T2
10	0x0012	TIMER2 OVF	Переповнення таймера/лічильника T2
11	0x0014	TIMER1 CAPT	Захоплення таймера/лічильника T1
12	0x0016	TIMER1 COMPA	Збіг A таймера/лічильника T1
13	0x0018	TIMER1 COMPB	Збіг B таймера/лічильника T1
14	0x001A	TIMER1 OVF	Переповнення таймера/лічильника T1
15	0x001C	TIMER0 COMPA	Збіг A таймера/лічильника T0
16	0x001E	TIMER0 COMPB	Збіг B таймера/лічильника T0
17	0x0020	TIMER0 OVF	Переповнення таймера/лічильника T0

Таблиця 5.2. – Вектори переривань за таймерами для ATmega2560

Номер вектору	Адреса	Джерело	Опис переривання
14	$001A	TIMER2 COMPA	Збіг A таймера/лічильника T2
15	$001C	TIMER2 COMPB	Збіг B таймера/лічильника T2
16	$001E	TIMER2 OVF	Переповнення таймера/лічильника T2
17	$0020	TIMER1 CAPT	Захоплення таймера/лічильника T1
18	$0022	TIMER1 COMPA	Збіг A таймера/лічильника T1
19	$0024	TIMER1 COMPB	Збіг B таймера/лічильника T1

Номер вектору	Адреса	Джерело	Опис переривання
20	$0026	TIMER1 COMPC	Збіг C таймера/лічильника T1
21	$0028	TIMER1 OVF	Переповнення таймера/лічильника T1
22	$002A	TIMER0 COMPA	Збіг A таймера/лічильника T0
23	$002C	TIMER0 COMPB	Збіг B таймера/лічильника T0
24	$002E	TIMER0 OVF	Переповнення таймера/лічильника T0
32	$003E	TIMER3 CAPT	Захоплення таймера/лічильника T3
33	$0040	TIMER3 COMPA	Збіг A таймера/лічильника T3
34	$0042	TIMER3 COMPB	Збіг B таймера/лічильника T3
35	$0044	TIMER3 COMPC	Збіг C таймера/лічильника T3
36	$0046	TIMER3 OVF	Переповнення таймера/лічильника T3
42	$0052	TIMER4 CAPT	Захоплення таймера/лічильника T4
43	$0054	TIMER4 COMPA	Збіг A таймера/лічильника T4
44	$0056	TIMER4 COMPB	Збіг B таймера/лічильника T4
45	$0058	TIMER4 COMPC	Збіг C таймера/лічильника T4
46	$005A	TIMER4 OVF	Переповнення таймера/лічильника T4
47	$005C	TIMER5 CAPT	Захоплення таймера/лічильника T5
48	$005E	TIMER5 COMPA	Збіг A таймера/лічильника T5
49	$0060	TIMER5 COMPB	Збіг B таймера/лічильника T5
50	$0062	TIMER5 COMPC	Збіг C таймера/лічильника T5
51	$0064	TIMER5 OVF	Переповнення таймера/лічильника T5

Для того щоб використовувати будь-які переривання треба підключити необхідну бібліотеку з іменами переривань

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

Для встановлення (заборонення) глобального дозволу необхідно встановити (скинути) біт загального дозволу переривань (Global Interrupt Enable bit) регістра стану ядра, який називається SREG (Status Register). Робити це рекомендується використовуючи макрос-функції:

```
sei();  //дозволити всі переривання
cli();  //заборонити всі переривання
```

Також необхідно включити відповідний біт або біти регістрів TIMSK*n* (дивись рисунок 5.1 або рисунок 5.2).

Дуже важливим кроком при використанні переривань є підготовка обробника переривань. Тобто необхідно написати функцію яка буде визиватися при виникненні той чи іншої події. Для того щоб компілятор зрозумів, що ця

функція і є обробник переривань необхідно виконати деякі умови. Загальні правила при написанні процедури обробки переривань наступні:

1) обробник переривання не може нічого приймати в якості аргументу, а також не може нічого повертати;
2) перед оголошенням обробника необхідно вказати, що він є процедурою обробки переривання;
3) обробник переривання повинен виконуватися, як можна швидше.

Зауважимо, що іноді необхідність у третьому правилі відпадає, найчастіше це трапляється тоді, коли практично вся обробка подій відбувається у перериваннях.

У середовищах розробки Atmel Studio, AVR Studio (AVR GCC), Arduino IDE опис обробника переривань можливий або перед функцією **main,** або після і уявляє собою наступну конструкцію:

```
ISR(SOURCE_vect)
{
// Тіло обробника переривання
}
```

де **SOURCE** це ім'я джерела вектору переривання згідно з таблицями 5.1, або 5.2.

Наприклад, нам потрібно написати обробник переривання при виникненні події переповнення таймера/лічильника Т5. У таблиці 5.2 знаходимо назву джерела TIMER5 OVF (вектор №51), дописуємо до нього суфікс **_vect**, а між словами в імені джерела переривань ставимо підкреслення:

TIMER5_OVF_vect

Тоді структура обробника цієї події буде виглядати як:

```
ISR(TIMER5_OVF_vect)
{
// Тіло обробника переривання
}
```

Зауважимо, якщо в тілі переривання змінюється значення деякої глобальної змінної, то вона повинна бути оголошена з модифікатором **volatile**.

В якості прикладу використання переривання по переповненню рахункового регістру таймер-лічильника напишемо програму яка буде змінювати стан піна A7 кожні 20,48 мілісекунд. Для цього задіймо п'ятий таймер-лічильник в звичайному режимі, тобто в режимі №0. Як було сказано раніше, формули (3.1) або (4.1), кожний такт таймеру триває $t_{такт} = N \cdot 62{,}5нс$. Врахуємо, що в цьому режимі таймер рахує від 0 до 65535, тобто робить 65536 тактів. Встановивши дільник рівним одиниці отримуємо, що таймер буде переповнюватись кожні

$$t = 65536 \cdot 62{,}5нс = 4{,}096мс. \quad (5.1)$$

Поділимо заданий час 20,48 мс на час переповнення таймеру 4,096 мс і знайдемо, що за вказаний час таймер переповниться 5 разів. Це означає, що в програмі потрібно рахувати кількість переповнень і як тільки буде досягнуте таке значення інвертувати стан відповідного піна. Таким чином маємо наступну програму:

Prog5.1

```
1    #define F_CPU 16000000UL    //  частота - 16MHz
2    #include <avr/io.h>         // Бібліотека вводу/виводу
3    #include <avr/interrupt.h>  // Бібліотека імен переривань
4    volatile uint8_t Counter=0;
5    int main(void){      //ATmega2560
6      DDRF |= (1<<DDF7);      //PF7=A7 як ВИХІД
7      //Режим NORMAL за замовчуванням
8      TCCR5B |= (1<<CS50);    //дільник N=1
9      TIMSK5 |= (1<<TOIE5);   //дозволяємо переривання по переповненню
10     sei();                  //дозволяємо глобальні переривання
11       while (1){};          //основна програма
12   }
13   ISR(TIMER5_OVF_vect){     //обробник переривання
14       Counter++;
15       if(Counter == 5){
16         PORTF ^= (1<<PORTF7);
17         Counter = 0;
18       }
19   }
```

Завантаживши **Prog5.1** отримуємо наступну часову діаграму.

Рисунок 5.3 - Часова діаграма після завантаження **Prog5.1**.

Проаналізувавши програму **Prog5.1** бачимо, що керувати часом переповнення таймеру можливо тільки налаштуванням відповідного дільника, а це означає, що час виклику обробника переривань по переповненню можна змінювати лише з великим кроком.

У цьому пункті ми підрахуємо мілісекунди, в деякому сенсі створимо аналог функції **`millis()`** з мови Wiring. Декларуємо змінну **`millisCounter`** яка буде збільшувати своє значення кожну мілісекунду. Так як змінна глобальна, то ми можемо зчитати її значення або змінити його в будь-якому місці програми і тому необхідно вказати модифікатор **`volatile`**. Виберемо режим CTC роботи таймеру та значення дільника 64. Частота з якою таймер буде досягати значення TOP у режимах CTC визначається формулою (4.5), тоді час досягнення TOP після скидання рахункового регістру розраховується як

$$t = \frac{T}{2} = \frac{1}{2 \cdot f_{OCnx}} = \frac{N \cdot (TOP + 1)}{f_{clk}}. \tag{5.2}$$

Нам треба, щоб час скидання таймеру відбувався кожну мілісекунду, тобто $t = 10^{-3} c$. Тоді, у нашому випадку, знаходимо

$$TOP + 1 = \frac{t \cdot f_{clk}}{N} = \frac{10^{-3} \cdot 16 \cdot 10^{6}}{64} = 250. \tag{5.3}$$

Prog5.2

```
1    #define F_CPU 16000000UL  // вкажемо компілятору частоту
2    #include <avr/io.h>         //Бібліотека вводу/виводу AVR
3    #include <util/delay.h>
4    #include <avr/interrupt.h>  // Бібліотека імен переривань
5
6    volatile uint32_t millisCounter=0;
7    int main(void) {
8      Serial.begin(9600);
9      TCCR5A =0;
10     TCCR5B |= (1<<WGM52)|(1<<WGM53);  //режим №12 CTC
11     TCCR5B |= (1 << CS51)|(1 << CS50);// дільник 64
12     ICR5 = 250-1;           // переривання виконується раз в мілісекунду
13     TIMSK5 |= (1<<OCIE5C); //Дозволяемо переривання TIMER5_COMPC_vect
14     sei();                  // Дозволяемо глобальні переривання
15
16     while (1) {
17       Serial.println (millisCounter);
18       _delay_ms(100);
19     }
20   }
21   ISR(TIMER5_COMPC_vect){
22            millisCounter++;
23   }
```

Звідси бачимо, що значення TOP яке необхідне помістити в порівняльний регістр дорівнює 250 – 1 = 249. Напишемо наступну програму в якій задіймо п'ятий таймер та канал С.

Цикл `while` (строки 16-19) в цій програмі можна було залишити порожнім, але ми для наочності зробили вивід значення змінної **`millisCounter`** в монітор порту скориставшись для простоти функціями мови Wiring **`Serial.`**`begin` (8-строка) и **`Serial.`**`println` (17-строка).

Зазначимо один дуже важливий аспект програми **Prog5.2**. У ній немає установки регістру OCR5C, і це значення дорівнює нулю. Після того як у регістрі TCNT5 значення досягає 249, воно скидається на нуль і в цей момент виявляється рівним зі значенням регістра OCR5C, а значить відбувається виклик переривання `TIMER5_COMPC_vect.`

Напишемо тепер програму, яка буде вимірюватиме час натискання на кнопку за допомогою переривання по захопленню. Режим захоплення означає, що при настанні деякої події, значення рахункового регістра TCNT*n* захоплюється регістром ICR*n* і зберігається в ньому до наступної події. Керування режимом захоплення здійснюється бітами ICNC*n*, ICES*n* регістра TCCR*n*B (див. рисунок 4.2).

За допомогою біта ICES*n* встановлюється активний фронт на піні ICP*n*. Якщо ICES*n* = 0, то активний негативний фронт сигналу, якщо ICES*n* = 1 – позитивний фронт.

Якщо біт ICNC*n* =0 (скинуто) схема придушення перешкод відключена і захоплення відбувається по першому активному фронту. Якщо біт ICNC*n* =1 (встановлено) схема придушення перешкод включена і захоплення відбувається по четвертому поспіль активному фронту.

При використанні переривання по захопленню необхідно встановити біт ICIE*n* регістра TIMSK*n* (див. рисунок 5.2).

Згідно з технічною документацією піни ICP*n* для мікроконтролера ATmega328: ICP1 = PB0 (D8);

та ATmega2560: ICP1 = PD4 (D-), ICP3 = PE7 (D-), ICP4 = PL0 (D49), ICP5 = PL1 (D48).

Для захоплення сигналу з виводу ICP*n*, цей пін повинен бути налаштований як ВХІД (розряд регістра DDR*x*, відповідний піну, повинен бути скинуто у «0»). Якщо ж він буде налаштований як ВИХІД, захоплення можна буде здійснювати програмно, керуючи відповідним розрядом порту.

Для виміру часу натискання кнопки будемо використовувати п'ятий таймер. Підключімо кнопку за наступною схемою (див. рисунок 5.4), де для боротьби з брязкотом контактів застосовано простіший RC фільтр. Також будемо вважати, що пін PL1 налаштовано на ВХІД і підтягнуте програмним засобам до +5 вольт. Тоді момент натискання кнопки буде відповідати негативному фронту, а відпускання – позитивному. Таким чином, нам потрібно виміряти час присутності на піні ICP5 логічного нуля. Для цього зробимо наступне:

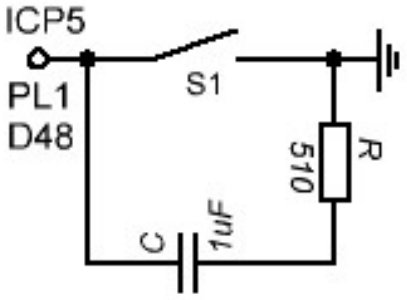


Рисунок 5.4 - Підключення кнопки з RC колом.

а) дозволимо запуск переривання по захопленню таймера командою

```
TIMSK5 |= (1<<ICIE5);
```

б) дозволимо глобальні переривання

```
sei();
```

в) опишемо обробник переривання по захопленню (див. таблицю 5.2)

```
ISR(TIMER5_CAPT_vect){
    //обробник переривання по захопленню
    }
```

Оголосимо 16-бітну без знакову змінну з іменем **`t_press`**. Ця змінна зберігатиме час тривалості натискання кнопки в тактах п'ятого таймера. Також оголосимо булеву змінну **`flag`**, яка дорівнює логічній одиниці, якщо мікроконтролер чекає натискання кнопки і логічному нулю, якщо натискання відбулося і йде процес вимірювання часу. Порахуємо в яких межах можна

виміряти час натискання при зроблених налаштуваннях. Так як дільник таймера включений на 1024 це означає, що кожен такт таймера буде проходити з частотою

$$16000000/1024 = 15625 \text{ Гц}. \quad (5.4)$$

Іншими словами, кожен такт таймера триває

$$1/15625 = 0{,}000064 \text{ с} = 64 \text{ мкс}. \quad (5.5)$$

Насправді 64 мікросекунди є мінімальний час, який можна виміряти, а також є точність, з якою ми вимірюємо час. Тобто не можна точно виміряти інтервал часу, тривалість якого не ділиться націло на 64. Наприклад, при вимірі інтервалу тривалістю 200 мкс ми отримаємо 64×3= 192 мкс. Для вимірювання часу натискання кнопки така точність цілком допустима, оскільки звичайне натискання кнопки триває більше 50мс = 50000мкс.

Розрахуємо тепер максимальну тривалість натискання кнопки, яку можна виміряти. Оскільки ми використовуємо 16-бітний таймер, то очевидно, що максимальна тривалість дорівнює часу переповнення таймера

$$64\text{мкс} \times 2^{16} = 64 \times 65536 = 4194304 \text{ мкс} = 4{,}194 \text{ с}, \quad (5.6)$$

тобто становить трохи більше чотирьох секунд.

Для того, щоб виміряти час натискання кнопки, нам потрібно спочатку вловити негативний фронт і одночасно з цим запустити процес рахунку, а потім зафіксувати позитивний фронт і зупинити рахунок. У програмі **Prog5.3** наведено повний лістинг коду.

Розглянемо обробник переривання рядки 38 – 49 програми **Prog5.3**. Ім'я вектору переривання TIMER5_CAPT_vect визначаємо з таблиці 5.2 для мікроконтролера ATmega2560. Спочатку роботи біт ICES5 скинуто (знаходиться у стані «0» за замовчуванням), що призводить до запуску переривання негативним фронтом, тобто у момент натискання кнопки. Також у цей момент значення змінної `flag` дорівнює `true`. Значить після проходження оператора `if` (рядок 39) виконуються рядки 40 - 42. На 40-му рядку скидається значення рахункового регістра першого таймера, на 41-му рядку змінної `flag` присвоюється значення `false`, на 42-му рядку встановлюється біт ICES5, що

Prog5.3

```
1    #define F_CPU 16000000UL  // вкажемо компілятору частоту
2
3    #include <avr/io.h>         // Бібліотека вводу/виводу AVR
4
5    #include <util/delay.h>     // Бібліотека затримок
6
7    #include <avr/interrupt.h>  // Бібліотека імен переривань
8
9    // кваліфікатор volatile інформує компілятор, що значення
10   // змінної буде змінюватися з зовні в невизначений час
11   volatile uint16_t t_press=0; //час натискання в тактах таймера
12   volatile bool flag = 1;      //1 чекаємо момент натискання
13                                //0 вимірюємо час натискання
14
15   int main(void) {
16     Serial.begin(9600);
17     DDRL &= ~(1<<DDL1); //пін PL1 як ВХІД (сюди підключається кнопка)
18     //ICP5=PL1=D48  захоплення по фронту на піні
19     PORTL |= (1<<PORTL1);// PULLUP
20
21     //WGM52=WGM51=WGM50=0 normal режим
22     //COM5A0=COM5A1=0     піни OC1A відключено
23     TIMSK5 |= (1<<ICIE5); //дозволяємо запуск переривання по захопленню
24     //ICNC5=0 придушник перешкод відключено
25     //ICES5=0 захоплення по негативному фронту
26
27     TCCR5B |= (1 << CS52)|(1 << CS50);//дільник 1024 и запуск таймеру
28     //16000000/1024=15625Гц або 1/15625=64 мікросекунди
29
30     sei();//дозволяємо глобальні переривання
31
32     while (1) {
33       Serial.println (t_press);
34       _delay_ms(1000);
35     }
36   }
37   //ISR — Interrupt Service Routine (обробник переривання)
38   ISR(TIMER5_CAPT_vect){
39       if (flag){
40        TCNT5=0;
41        flag=0;
42        TCCR5B |= (1 << ICES5); // захоплення по позитивному фронту
43       }
44       else{
45        t_press = ICR5;
46        flag=1;
47        TCCR5B &= ~(1 << ICES5); // захоплення по негативному фронту
48       }
49   }
```

призведе до наступного запуску обробника переривання в момент позитивного фронту тобто коли кнопка відпускається. При наступі цієї події знову викликається обробник TIMER5_CAPT_vect. Але зараз `flag` дорівнює `false` і, отже, буде виконано рядки 44 – 48. На 44-му рядку відбувається зчитування

значення регістру захоплення у змінну `t_press`. Це і є час утримання кнопки в тактах таймера. У рядках 46, 47 відбувається підготовка до наступного виміру.

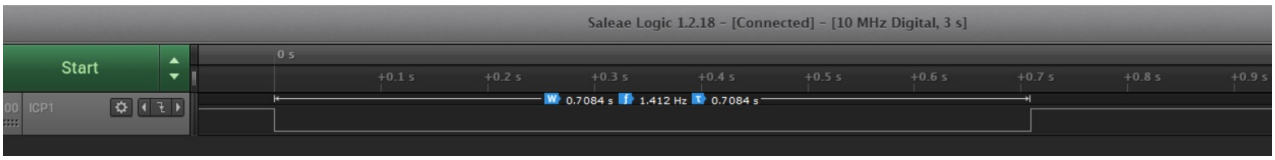


Рисунок 5.5 - Тривалість натискання кнопки в логічному аналізаторі.

Проведемо порівняння вимірювання тривалості натискання двома способами використовуючи програму **Prog5.3** та логічний аналізатор (його потрібно налаштувати на запуск по негативному фронту), підключений до кнопки. Отримуємо що програма видає значення, представлені на рисунку 5.6, а аналізатор на рисунку 5.5.

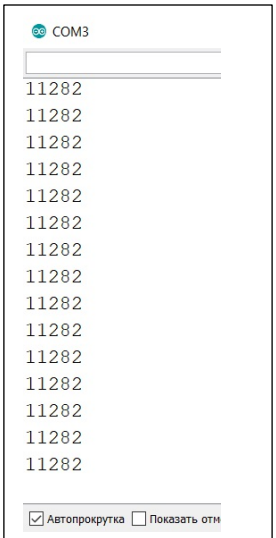


Рисунок 5.6 Тривалість натискання в моніторі порту.

Аналізатор показує, що натискання кнопки тривало 0,708 секунд. Програма виводить значення 11282 помножуючи це число на 0,064 мілісекунди отримуємо 11282 × 0,064 = 722 мілісекунд. Тобто розбіжність не перевищує 2-х відсотків.

Зауважимо, що програма не є «дурнястійкою», тобто при натисканні на кнопку більш ніж 4,2 секунд отримаємо не коректний результат.

Підкреслимо той факт, що вся основна робота по вимірюванню тривалості натискання кнопки проводиться в обробнику переривань. Основна програма (строки 32 – 35) задіяні тільки для того щоб вивести отриманий результат на комп'ютері в монітор порту. Для цього у даному випадку заради простоти використовуються «ардуінівські» функції **`Serial.`**`begin(9600)` (строка 16) та **`Serial.`**`println ()` (строка 33). Ці функції дуже спрощують роботу по протоколу UART і не впливають на розуміння основних ідей пов'язаних з механізмами роботи переривань за таймерами.

Також слід зазначити, що якби ми захотіли виводити результати вимірювань тривалості натискання кнопки у секундах, то нам довелося б використовувати

тип float який для мікроконтролерів AVR дуже «тяжкий». Це означає, що перерахунок тактів таймеру в секунди потрібно було б робити в тілі основної програми, а не як не в обробнику переривань.

Хід роботи

Завдання №5.1.

Використавши переривання по переповненню відповідного таймеру в нормальному режимі налаштуйте на вказаному піні сигнал тривалістю t_1 високого рівня та t_2 – низького рівня. Поясніть з якою точністю це теоретично можна зробити. Перевірте розрахунки експериментально.

Таблиця 5.3. – Вхідні данні для завдання 5.1

Варіант	№ таймеру, дільник	t_1, t_2	Пін
1	T0, 1	100 мкс, 150 мкс	A5
2	T2, 8	1,5 мс; 2,9 мс	A4
3	T0, 64	9,2 мс; 14 мс	A3
4	T2, 256	32,5 мс; 12,1 мс	A2
5	T0, 1024	48 мс; 80,5 мс	A1
6	T2, 1	250 мкс, 200 мкс	A0
7	T0, 8	3,1 мс; 2,0 мс	A5
8	T2, 64	12 мс 7,1 мс	A4
9	T0, 256	24,5 мс; 53 мс	A3
10	T2, 32	3,2 мс; 2,1 мс	A2
11	T2, 64	4,1 мс; 17,5 мс	A1
12	T0, 1	300 мкс; 350 мкс	A0
13	T2, 128	14,2 мс; 8,1 мс	A5
14	T0, 64	6,1 мс; 11 мс	A4
15	T2, 256	19,7 мс; 35 мс	A3
16	T0, 1024	62,8 мс; 15,9мс	A2
17	T2, 1	400 мкс, 250 мкс	A1
18	T0, 8	1,8 мс; 3,5 мс	A0
19	T2, 64	16 мс; 13,2 мс	A5
20	T0, 256	28,3 мс; 8,5 мс	A4

Завдання №5.2.

За допомогою переривання за збігом на першому таймері (або іншому по вказанню викладача) зробити так щоб значення глобальній змінній інкрементувалося на одиницю через кожні *t* мікросекунд, також через цей час стан піна повинен інвертуватися. Конкретні значення *t*, дільника, переривання та номер піна вибрати згідно зі своїм варіантом з таблиці 5.4. Провести розрахунки теоретично та перевірити отримані результати за допомогою логічного аналізатора.

Таблиця 5.4. – Вхідні данні для завдання 5.2

Варіант	Дільник	Переривання, № режиму	*t,* мкс	Пін
1	256	COMPB, №12	10000	A0
2	1	COMPA, №4	1500	A1
3	8	COMPB, №12	5000	A2
4	1024	COMPA, №4	40000	A3
5	256	COMPB, №12	12000	A4
6	1	COMPA, №4	625	A5
7	8	COMPB, №12	3000	D2
8	1024	COMPA, №4	80000	D7
9	256	COMPB, №12	8000	D4
10	1	COMPA, №4	2000	D8
11	8	COMPB, №12	2500	D1
12	1024	COMPA, №4	1920	A0
13	256	COMPB, №12	14000	A1
14	1	COMPA, №4	500	A2
15	8	COMPB, №12	1500	A3
16	1024	COMPA, №4	5120	A4
17	256	COMPB, №12	6000	A5
18	1	COMPA, №4	750	D2
19	8	COMPB, №12	1200	D7
20	1024	COMPA, №4	8320	D4

Завдання №5.3.

Зробіть наступне.

1. Налаштуйте пін ICP1=D8 для ATmega328 або ICP4=D49 для ATmega2560 на вихід та сформуйте на ньому меандр будь-яким програмним засобом в фоновому режимі відповідної частоти за допомогою 8-бітного таймеру.
2. Переконайтеся за допомогою логічного аналізатору, що на необхідному піні присутній потрібний сигнал.
3. Змініть програму з пункту 1 таким чином, щоб вона додатково вимірювала частоту сигналу на відповідному піні за допомогою режиму захвату. Для виводу значення отриманої частоти в монітор порту можна скористатись «ардуінівськими» функціями.
4. Розрахувати абсолютну та відносну точності отриманих результатів.

Таблиця 5.5. – Вхідні данні для завдання 5.3

Варіант	Пін*	Частота меандру,Гц
1	ICP1	5000
2	ICP1	31250
3	ICP1	25000
4	ICP1	6250
5	ICP1	10000
6	ICP1	15625
7	ICP1	12500
8	ICP1	50000
9	ICP1	3125
10	ICP1	1250
11	ICP4	625
12	ICP4	31250
13	ICP4	6250
14	ICP4	15625
15	ICP4	50000
16	ICP4	1250
17	ICP4	5000
18	ICP4	25000
19	ICP4	10000
20	ICP4	12500

* - якщо викладач не задав інший мікроконтролер

Лабораторна робота №6

Зовнішні переривання

Мета: навчитися теоретично розраховувати та експериментально визначати параметри роботи зовнішніх переривань мікроконтролерів ATmega328 і ATmega2560 в різних режимах роботи.

Теоретичні відомості

Першим кроком розглянемо які згідно з технічної документації існують зовнішні переривання. В таблиці Interrupt Vectors in ATmega328 (Table 16-1) для мікроконтролера ATmega328 та в таблиці Interrupt Vectors in ATmega640/1280/1281/2560/2561 (Table 32) для мікроконтролера ATmega2560 знаходимо наступну інформацію (див. таблицю 6.1 та 6.2).

Таблиця 6.1. – Вектори зовнішніх переривань для ATmega328

Номер вектору	Адреса	Джерело	Опис переривання
2	0x0002	INT0	Зовнішнє переривання 0
3	0x0004	INT1	Зовнішнє переривання 1
4	0x0006	PCINT0	Переривання 0 при зміні стану виводів
5	0x0008	PCINT1	Переривання 1 при зміні стану виводів
6	0x000A	PCINT2	Переривання 2 при зміні стану виводів

Таблиця 6.2. – Вектори зовнішніх переривань для ATmega2560

Номер вектору	Адреса	Джерело	Опис переривання
2	$0002	INT0	Зовнішнє переривання 0
3	$0004	INT1	Зовнішнє переривання 1
4	$0006	INT2	Зовнішнє переривання 2
5	$0008	INT3	Зовнішнє переривання 3
6	$000A	INT4	Зовнішнє переривання 4
7	$000C	INT5	Зовнішнє переривання 5
8	$000E	INT6	Зовнішнє переривання 6
9	$0010	INT7	Зовнішнє переривання 7
10	$0012	PCINT0	Переривання 0 при зміні стану виводів
11	$0014	PCINT1	Переривання 1 при зміні стану виводів
12	$0016	PCINT2	Переривання 2 при зміні стану виводів

Спочатку розглянемо зовнішні переривання INT*n*.

Для дозволу цих переривань існує регістр EIMSK (External Interrupt Mask Register). Його структура у мікроконтролері ATmega328P зображена на рисунку 6.1.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 6.1 - Структура регістра EIMSK мікроконтролера ATmega328.

Структура той самого регістра але мікроконтролера ATmega2560 зображена на рис.6.2.

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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	

Рисунок6.2 - Структура регістра EIMSK мікроконтролера ATmega2560.

З рисунків 6.1 та 6.2 зрозуміло, що в разі потреби використати те чи інше переривання необхідно встановити відповідний біт у регістрі EIMSK.

З технічної документації знаходимо до яких пінів мікроконтролерів належать зовнішні переривання INT*n*.

Таблиця 6.3. – Зв'язок між пінами та зовнішніми перериваннями INT*n*.

INT*n*	ATmega328		ATmega2560	
INT0	PD2	D2	PD0	D21
INT1	PD3	D3	PD1	D20
INT2	-		PD2	D19
INT3	-		PD3	D18
INT4	-		PE4	D2
INT5	-		PE5	D3
INT6	-		PE6	-
INT7	-		PE7	-

Для остаточного налаштування кожного зовнішнього переривання INT*n* потрібно два біти ISC*n*0 і ISC*n*1 регістрів EICRA і EICRB (**External Interrupt Control Register**). Таким чином для налаштування INT0 і INT1 мікроконтролера ATmega328 використовується чотири біта регістра EICRA (див. рисунок 6.3), а

для налаштування всіх переривань INT*n* мікроконтролера ATmega2560 потрібно задіяти вісім біт регістра EICRA та вісім біт регістра EICRB (див. рисунок 6.4).

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 6.3 - Регістр EICRA мікроконтролера ATmega328.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	
Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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	

Рисунок 6.4 - Регістри EICRA і EICRB мікроконтролера ATmega2560.

Розберемося які можливі події можна відстежувати за допомогою зовнішніх переривань INT*n* налаштувавши відповідно біти ISC*n*0 і ISC*n*1. Для цього подивимось пункт 17.2.1 технічної документації на ATmega328, або таблицю 34 для ATmega2560. Інформація з цих документів приведена в таблиці 6.4.

Таблиця 6.4. – Налаштування типу події для зовнішніх переривань INT*n*.

ISC*n*1	ISC*n*0	Подія
0	0	наявність сигналу низького рівня
0	1	будь-яка зміна сигналу
1	0	зміна сигналу від високого рівня до низького (негативний фронт)
1	1	зміна сигналу від низького рівня до високого (позитивний фронт)

З таблиці 6.4 бачимо, що доступні такі події на відповідному піні які призведуть до виклика потрібного обробника переривання: низький сигнал, будь-який фронт, негативний фронт або позитивний фронт.

Використаємо отримані данні для створення простішого проекту.

Будемо керувати вбудованим світлодіодом на піні PB7=D13 мікроконтролера ATmega2560 за допомогою кнопки підключеною до піна який пов'язано з зовнішнім перериванням INT5 (див. таблицю 6.3), тобто пін PE5=D3. При натисканні на кнопку має викликатися обробник переривання INT5 який буде інвертувати стан піна PB7. Для боротьби з брязкотом контактів кнопці використаємо RC фільтр (див. рисунок 6.5). Ця схема підключення кнопки вимагає програмного підтягування піна до +5 вольт і тоді при натисканні кнопки на цьому піні з'явиться логічний нуль. Зауважимо, що номінали як резистора, так і конденсатора можна міняти приблизно на порядок.

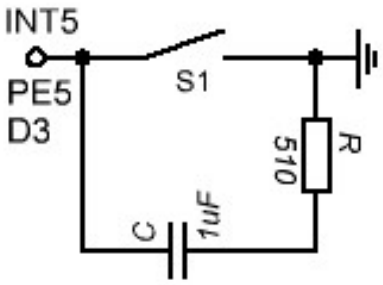


Рисунок 6.5 - Підключення кнопки

Таким чином маємо наступну програму

Prog6.1

```
1    #define F_CPU 16000000UL    // Частота - 16MHz
2    #include <avr/io.h>         // Бібліотека вводу/виводу AVR
3    #include <avr/interrupt.h>  // Бібліотека імен переривань
4
5    int main(void) {
6     DDRB |= (1<<DDB7);     // PB7=D13 ATmega2560 як ВИХІД
7     DDRE &= ~(1<<DDE5);    // PE5=D3 як ВХІД (сюди підключається кнопка)
8     PORTE |= (1<<PORTE5);  // к PE5 підтягуємо до + живлення
9
10    EICRB |= (1 << ISC51); // ISC51=1 - переривання по FALLING на INT5
11    EIMSK |= (1 << INT5);  // Дозволяємо переривання INT5
12    sei();                 // Дозволяємо глобальні переривання
13    while(1) {}            // Основна програма
14   }
15
16   //обробник переривання INT5
17   ISR(INT5_vect) {
18    PORTB ^= (1<<PORTB7);  //інвертуємо стан піна PB7=D13
19   }
```

Зараз розглянемо зовнішні переривання (**Pin Change Interrupts**) PCINT*n*.

Основна відмінність цих переривань від INT*n* полягає в тому, що вони завжди викликаються при будь-якій зміні стану на відповідних пінах і це не підлягає налаштуванню. Згідно з таблиць 6.1 та 6.2 таких переривань три як в мікроконтролері ATmega328 так і в ATmega2560. Інша дуже важлива відмінність переривань PCINT*n* пов'язана з тим, що кожне переривання поєднує декілька (як

правило вісім) пінів. Зв'язок між перериваннями та пінами представлено в таблиці 6.5.

Таблиця 6.5. – Відповідність пінів і переривань PCINT*n*.

	Піни PCINT*n*	ATmega328P	ATmega2560
	PCIE0		
Переривання PCINT0	PCINT0	PB0 D8	PB0 D53
	PCINT1	PB1 D9	PB1 D52
	PCINT2	PB2 D10	PB2 D51
	PCINT3	PB3 D11	PB3 D50
	PCINT4	PB4 D12	PB4 D10
	PCINT5	PB5 D13	PB5 D11
	PCINT6	PB6 -	PB6 D12
	PCINT7	PB7 -	PB7 D13
	PCIE1		
Переривання PCINT1	PCINT8	PC0 A0	PE0 D0
	PCINT9	PC1 A1	PJ0 D15
	PCINT10	PC2 A2	PJ1 D14
	PCINT11	PC3 A3	PJ2 -
	PCINT12	PC4 A4	PJ3 -
	PCINT13	PC5 A5	PJ4 -
	PCINT14	PC6 RESET	PJ5 -
	PCINT15	-	PJ6 -
	PCIE2		
Переривання PCINT2	PCINT16	PD0 D0	PK0 A8
	PCINT17	PD1 D1	PK1 A9
	PCINT18	PD2 D2	PK2 A10
	PCINT19	PD3 D3	PK3 A11
	PCINT20	PD4 D4	PK4 A12
	PCINT21	PD5 D5	PK5 A13
	PCINT22	PD6 D6	PK6 A14
	PCINT23	PD7 D7	PK7 A15

З таблиці 6.5 бачимо, що, наприклад, переривання PCINT0 може бути викликане при будь якій зміні стану на пінах від PB0 до PB7. Також з цієї таблиці бачимо, що для того щоб дозволити переривання PCINT*n* потрібно встановити відповідний біт PCIE*n*. Ці біти знаходяться в регістрі PCICR (**Pin Change Interrupt Control Register**) структура якого зображена на рисунку 6.6.

Bit	7	6	5	4	3	2	1	0	
			–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок6.6 - Структура регістра PCICR для ATmega328 та ATmega2560.

Ще одна корисна інформація з таблиці 6.5 є та, що в ній показано які піни пов'язані з перериваннями PCINT*n* можливо використовувати на платформах Arduino UNO та Arduino Mega. Так, наприклад, ми не можемо на платформі Arduino UNO задіяти піни PCINT6, PCINT7 та PCINT14.

Регістри PCMSK0, PCMSK1 і PCMSK2 (**Pin Change Mask Register**) використовуються для того щоб вказати яким конкретно пінам дозволено генерувати сигнал запиту переривання. Для цього досить встановити потрібний біт у відповідному регістрі. На рисунку 6.7 зображені регістри PCMSK0, PCMSK1 та PCMSK2 мікроконтролера ATmega2560. Відмінність таких же регістрів, але для мікроконтролера ATmega328 полягає тільки в тому, що пін PCINT15 у цього мікроконтролера відсутній.

Bit	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	
Bit	7	6	5	4	3	2	1	0	
	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
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	
Bit	7	6	5	4	3	2	1	0	
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

Рисунок6.7 - Регістри PCMSK0, PCMSK1, PCMSK2 мікроконтролера ATmega2560.

Після отримання теоретичних знань перейдемо до практичного використання. Підключимо дві кнопки до переривання PCINT0. Причому будемо використовувати піни PCINT0 та PCINT1 (див. рисунок 6.8). Можлива не велика плутанина, яка полягає в тому, що виробник мікроконтролерів дав одне і те саме ім'я як перериванням так і пінам, але уважне читання дозволить зрозуміти де мова йдеться про переривання, а де про піни.

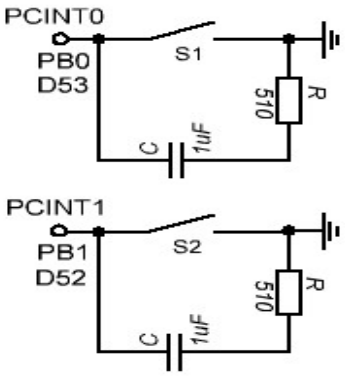


Рисунок 6.8 - Підключення кнопок.

Згідно таблицям 6.1, 6.2 обробник потрібного нам переривання викликається конструкцією

```
ISR(PCINT0_vect)
{
// Тіло обробника переривання
}
```

Для дозволу цього переривання треба встановити біт PCIE0 регистра PCICR, тобто потрібна команда:

```
PCICR |= (1 << PCIE0);
```

Так як дві кнопки підключені до пінів PCINT0 та PCINT1, то це означає, що нам потрібно дозволити саме цим пінам генерувати переривання PCINT0. Це робиться встановленням бітів PCINT0 та PCINT1 регістру PCMSK0 (див. рисунок 6.7). Тобто необхідна команда:

```
PCMSK0 |= (1 << PCINT0)|(1 << PCINT1);
```

Напишемо зараз програму яка буде двома кнопками керувати одним вбудованим світлодіодом на D13. Будемо використовувати ATmega2560, тоді D13 = PB7. Будемо вважати, що кнопка №1 підключена до піну PB0, а кнопка №2 – до PB1. При натисканні на першу кнопку світлодіод має включитися, а при натисканні на другу – виключитися. Так як обидві кнопки використовують переривання PCINT0, то це переривання буде викликатися кожного разу при

натисканні або відпусканні якої завгодно кнопки. Відслідкувати апаратними засобами яка з кнопок була натиснута неможливо. Тому потрібно в обробнику переривання написати такий код, який зможе визначити, по-перше, що кнопка була натиснута, а не відпущена, а по-друге, яку саме кнопку натискали.

Prog6.2

```
1                          // ATmega2560
2
3    #define F_CPU 16000000UL    // Частота - 16MHz
4    #include <avr/io.h>         // Бібліотека вводу/виводу AVR
5    #include <avr/interrupt.h>  // Бібліотека імен переривань
6
7    int main() {
8     DDRB |= (1<<DDB7);     // Пін D13=PB7 ATmega2560 як ВИХІД
9     PORTB &= ~(1<<PORTB7); // викл
10
11    DDRB &= ~((1<<DDB0)|(1<<DDB1)); //D53=PB0,D52=PB1 на ВХОД
12    PORTB |= (1<<PORTB0)|(1<<PORTB1); // підтягуємо до + живлення
13    //D53 - кнопка1;    D52 - кнопка2
14
15    PCICR |= (1 << PCIE0); // Дозволяємо переривання групи PCINT0
16    // Дозволяємо переривання на пінах PCINT0=PB0, PCINT1=PB1
17    PCMSK0 |= (1 << PCINT0)|(1 << PCINT1);
18    sei();                 // Дозволяємо глобальні переривання
19    while(1) {}            // Основна програма
20   }
21
22   // обробник переривання PCINT0
23   ISR(PCINT0_vect) {
24     if( !(PINB & (1<<PINB0))) PORTB |=(1<<PORTB7);
25     if( !(PINB & (1<<PINB1))) PORTB &=~(1<<PORTB7);
26   }
```

Розглянемо як працює програма **prog6.2.**

На строках №8 та 9 налаштовується пін PB7 = D13 для використання вбудованого світлодіоду. На строчці № 8 пін PB7 визначається як ВИХІД, а команда на строчці №9 виключає вбудований світлодіод.

На строках №11 та 12 йде підготовка пінів до підключення кнопок. Строчка №11 налаштовує піни PB0 та PB1 як ВХОДИ. Команда на строчці №12 програмно підтягує вказані піни до «плюса» живлення.

В строках 15 – 18 налаштовуються дозволи до запуску переривань. На строчці 15 встановлюється дозвіл для переривання PCINT0. Строчка 17 дозволяє виклик переривання PCINT0 при будь-якому фронті на пінах PCINT0 та PCINT1. На строчці 18 встановлюється глобальний дозвіл для всіх переривань.

Для того щоб зрозуміти логіку роботи обробника переривань розберемо наступні моменти. При натисканні кнопки яка підключена до PB0 регістр PINB приймає наступне значення:

```
PINB=0bxxxxxx10.
```

Чому так відбувається? Тому що дві кнопки підключені до пінів які підтягнути до «плюсу», це означає, що коли кнопки не натиснути на цих пінах знаходиться логічна одиниця. Коли на кнопку відбувається натискання, то на відповідному піні з'являється логічний нуль. Про решту бітів нам нічого не відомо і їх стан нам не важливий. Аналогічно, якщо відбувається натискання на кнопку яка підключена до PB1, то регістр PINB приймає значення:

```
PINB=0bxxxxxx01.
```

Таким чином при натисканні кнопки на відповідному піні з'являється негативний фронт, що призводить до виклику обробника переривань за вектором **`PCINT0_vect`**.

Врахуємо, що

```
(1<<PINB0)= 00000001
```

та

```
(1<<PINB1)= 00000010.
```

Звідси знаходимо, що у випадку натискання на кнопку яка підключена до PB0 вираз `PINB & (1<<PINB0)` приймає значення

```
  0bxxxxxx10
& 0b00000001
  ----------
  0b00000000
```

яке дорівнює нулю, тобто відповідає логічному *false*. На це значення на строчці 24 діє оператор логічного НІ (окличний знак) і перетворює його в *true*. Це призводить до того, що оператор **`if`** на 24 строчці виконується і відбувається включення світлодіоду який підключено до піну PB7.

Таким же самим чином розглянувши вираз `PINB & (1<<PINB1)` у випадку натискання на кнопку яка підключена до піна PB0 отримуємо, що він дорівнює логічній одиниці. Так відбувається тому, що компілятор кожне ціле не нульове число в мові C або C++ сприймає як логічне *true*.

Отже маємо:

```
  0bxxxxxx10
& 0b00000010
  ----------
  0b00000010
```

яке відповідає логічній одиниці. Після дії логічного НІ значення перетворюється в *false*, а це призводить до того, що оператор на 25 строчці не виконується.

У випадку натискання на кнопку яка підключена до PB1 вираз `PINB & (1<<PINB0)` дорівнює

```
  0bxxxxxx01
& 0b00000001
  ----------
  0b00000001
```

Це значення відповідає логічному *true* і після дії оператора НІ перетворюється в *false*. Таким чином оператор **`if`** на 24 строчці не виконується.

Навпаки, вираз `PINB & (1<<PINB1)` у випадку натискання на туж саму кнопку приймає значення

```
  0bxxxxxx01
& 0b00000010
  ----------
  0b00000000
```

яке відповідає логічному *false* і після дії оператора НІ перетворюється в *true*, тобто оператор на 25 строчці зараз виконається.

Підсумувавши вищесказане можна зробити висновок, що не має апаратних можливостей відрізнити події які викликають зовнішнє переривання PCINT*n*. Але можна таким чином написати обробник переривання використавши програмні засоби, що мікроконтролер буде відрізняти різні події які призводять до виклику переривання.

Для більшого зрозуміння спробуйте пояснити, що відбувається при відпусканні першої та другої кнопок.

Хід роботи

Завдання 6.1.

Підключити три світлодіоди та кнопку (див. рисунок 6.9). Піни Dx, Dy, Dz та Px вибрати згідно свого варіанту (див. таблицю 6.6). Зобразити монтажну схему підключення. Написати програму, задіяв відповідне зовнішнє переривання, таким чином щоб при кожному натисканні або відпусканні кнопці (див. таблицю 6.6) включався наступний світлодіод, попередній при цьому має виключитися. При спрацюванні кнопки чотири рази всі світлодіоди мають бути вимкнені. Потім все повторюється знову. Поясніть у якому випадку запис логічної одиниці або логічного нуля у відповідний біт вмикає або вимикає світлодіод.

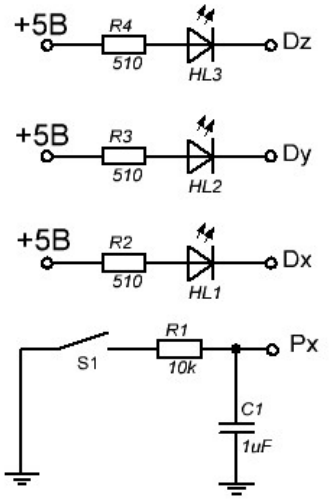


Рисунок 6.9 - Підключення світлодіодів та кнопки.

Таблиця 6.6. – Вхідні данні для завдання 6.1

Варіант	Px, МК	Кнопка	Dx	Dy	Dz
1	INT0, 328	натискання	A0	D3	D8
2	INT1, 328	натискання	A1	D11	D9
3	INT0, 2560	натискання	A2	D3	D10
4	INT1, 2560	натискання	A3	D4	D11
5	INT0, 328	відпускання	A4	D5	D12
6	INT1, 328	відпускання	A5	D6	D13
7	INT0, 2560	відпускання	A0	A1	D7
8	INT1, 2560	відпускання	A1	A2	D8
9	INT0, 328	натискання	A2	A3	D9
10	INT1, 328	натискання	A3	A4	D10
11	INT0, 2560	натискання	A4	A5	D11
12	INT1, 2560	натискання	A5	A0	D12
13	INT0, 328	відпускання	D11	D5	A1
14	INT1, 328	відпускання	D1	D6	A2
15	INT0, 2560	відпускання	D4	D7	A3
16	INT1, 2560	відпускання	D6	D8	A4
17	INT0, 328	натискання	D7	D9	A5
18	INT1, 328	відпускання	D8	D10	D13
19	INT0, 2560	натискання	D9	D11	D12
20	INT1, 2560	відпускання	D10	D12	A5

Завдання №6.2.

Підключити світлодіоди та кнопки за схемою зображеною на рисунку 6.10. Написати програму щоб при натисканні на першу кнопку змінювався стан першого світлодіоду, а при натисканні на другу – другого. Причому необхідно використовувати відповідне зовнішнє переривання PCINT яке відповідає пінам до яких підключені кнопки. Піни Px, Py та Dx, Dy виберіть згідно свого варіанту (див. таблицю 6.7). Тип мікроконтролера вказує викладач. Зобразити монтажну схему підключення.

Завдання 2+ (на додаткові бали).
Змініть програму таким чином, щоб світлодіоди змінювали свій стан при відпусканні відповідній кнопки. Детально опишіть алгоритм роботи цієї програми.

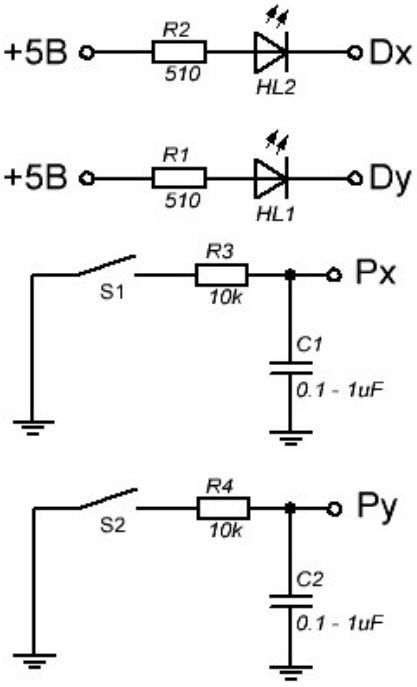


Рисунок 6.10 - Підключення світлодіодів та кнопок.

Таблиця 6.7. – Вхідні данні для завдання 6.2

Варіант	Px	Py	Dx	Dy
1	PCINT2	PCINT5	A1	D3
2	PCINT3	PCINT4	A2	D4
3	PCINT8	PCINT10	A3	D5
4	PCINT10	PCINT9	A4	D6
5	PCINT16	PCINT20	A5	A1
6	PCINT17	PCINT21	A0	A2
7	PCINT18	PCINT22	A1	A3
8	PCINT19	PCINT23	A2	A4
9	PCINT4	PCINT0	A3	A5
10	PCINT5	PCINT1	A4	A0
11	PCINT20	PCINT17	A5	D5
12	PCINT21	PCINT18	D0	D6
13	PCINT22	PCINT19	D1	D7
14	PCINT23	PCINT16	D4	A2
15	PCINT0	PCINT2	D6	A3
16	PCINT1	PCINT3	D7	D10
17	PCINT8	PCINT9	A4	D11
18	PCINT5	PCINT0	A5	D12
19	PCINT4	PCINT1	D10	D5
20	PCINT3	PCINT2	A0	D7

Завдання №6.3.

Підключити кнопку за схемою зображеною на рисунку 6.11. Використавши відповідне піну переривання PCINT написати програму яка буде вимірювати час натискання кнопки, причому необхідно задіяти 16-бітний таймер в нормальному режимі. Організувати вивід часу натискання кнопки в мілісекундах в монітор порту скориставшись для простоти «ардуінівськими» функціями. Опишіть детально як працює ваша програма. Перевірити отриманий час натискання кнопки за допомогою логічного аналізатору. Зобразити монтажну схему підключення.

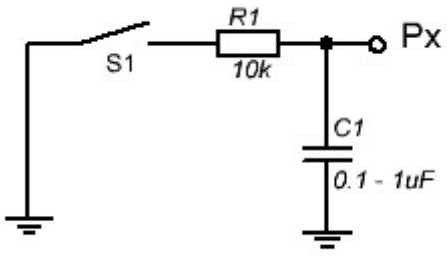


Рисунок 6.11 - Підключення кнопки з RC колом.

Підказка: поєднайте програми **Prog5.3** та **Prog6.2**.

Таблиця 6.8. – Вхідні данні для завдання 6.3

Варіант	Пін	Варіант	Пін
1	PCINT16	11	PCINT10
2	PCINT17	12	PCINT0
3	PCINT18	13	PCINT1
4	PCINT19	14	PCINT2
5	PCINT20	15	PCINT3
6	PCINT21	16	PCINT4
7	PCINT22	17	PCINT5
8	PCINT23	18	PCINT18
9	PCINT8	19	PCINT22
10	PCINT9	20	PCINT19

Лабораторна робота №7

Інтерфейс UART

Мета: навчитися налаштовувати апаратний інтерфейс UART у мікроконтролерах ATmega328 та ATmega2560 на прикладі роботи з термінальною програмою Serial Monitor (Arduino IDE) або Virtual Terminal (Proteus).

Теоретичні відомості

У більшості мікроконтролерів сімейства AVR інтерфейс UART (USART) підтримується на апаратному рівні. Термін USART означає: Universal Synchronous Asynchronous Receiver Transmitter. Тобто універсальний синхронно-асинхронний прийомо-передавач. Іншими словами, за допомогою цього інтерфейсу можна приймати дані з іншого пристрою або передавати їх на нього. Зазвичай, у практичних задачах використовують UART інтерфейс, тобто асинхронний спосіб прийому/передачі даних. Його ми і розглянемо. Для пов'язування мікроконтролера з комп'ютером в основному використовується перетворювач UART – USB, який іноді називають UART – TTL або UART – COM перетворювачем. За його допомогою, USB роз'єм комп'ютера поєднується з Rx і Tx пінами мікроконтролера (див. рисунок 7.1).

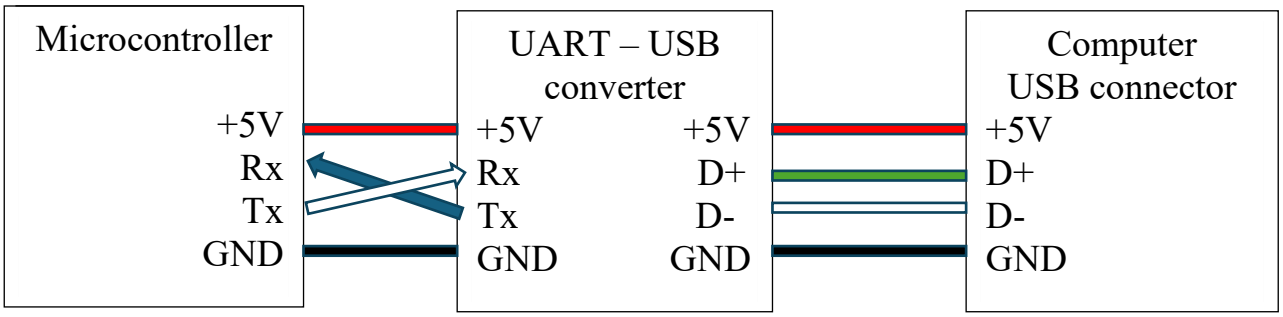


Рисунок 7.1 - З'єднання комп'ютера з мікроконтролером.

По каналу Rx (Receiver) здійснюється прийом даних, а по каналу Tx (Transmitter) – передача. Прийом і передача можуть здійснюватися одночасно, тому цей інтерфейс є повнодуплексним зв'язком.

Так як ми користуємось платформами Arduino UNO і MEGA, то перетворювач UART – USB в них вже встановлений. Зазвичай це мікросхема

ATMEGA16U2 на оригінальних платах Arduino та CH340G на копіях. Тому додаткове обладнання нам не знадобиться.

Розглянемо принцип організації протоколу UART. Потік даних, що передаються по цьому протоколу являє собою набір кадрів. Кожен кадр містить:

- стартовий біт;
- від 5 до 9 бітів даних;
- стоповий біт.

Перед стоповим бітом може бути біт парності: парний або непарний.

На боці комп'ютера необхідно мати термінальну програму, яка буде зчитувати дані, що відправляє мікроконтролер та посилати свої дані на нього. Таких програм існує багато, ми скористаємось вбудованою у Arduino IDE термінальною програмою, яка має назву Serial Monitor і викликається комбінацією Ctrl+Shift+M.

Розглянемо структуру кадру (див. рисунок 7.2) з офіційної документації на мікроконтролер.

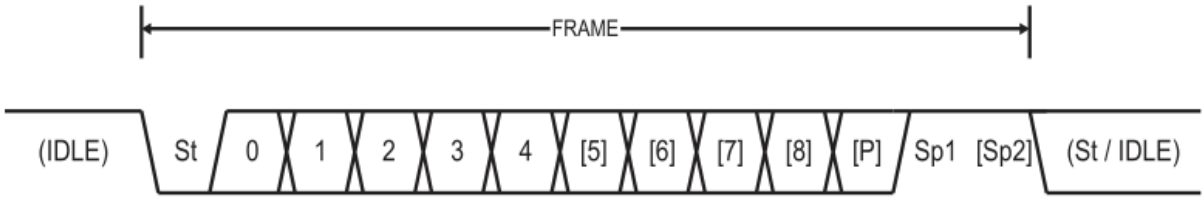


Рисунок 7.2 - Структура кадру UART.

Поки інформація не передається (IDLE – холостий хід), лінія знаходиться у високому стані, тобто на ній присутня логічна одиниця. Як тільки передаючий пристрій збирається надіслати інформацію, він встановлює на лінії логічний нуль. Таким чином передається стартовий біт (St). Потім передаються інформаційні біти. Якщо передається одиниця, то на лінії встановлюється високий рівень, якщо нуль – низький. Кількість інформаційних бітів може бути від 5 до 9 (про це треба домовитись перед з'єднанням пристроїв). Після завершення передачі інформаційних бітів, може йти біт парності (про його наявність і тип також має бути домовлено перед початком роботи). Завершує

передачу стоп-біт, який завжди має високий логічний рівень, а його тривалість може дорівнювати 1, 1,5 або 2 тривалості передачі одного біта.

Тривалість передачі біта пов'язана зі швидкістю роботи UART протоколу. Зазвичай вона рівна 9600 бод, це означає, що час передачі біта буде рівний

$$\frac{1}{9600} \approx 1{,}04 \cdot 10^{-4} сек = 104 мкс .$$

При описі протоколу UART зазвичай використовується короткий запис:

ЦИФРА-БУКВА-ЦИФРА.

Де:

ЦИФРА вказує на кількість інформаційних біт (від 5 до 9), які передаються у кадрі

БУКВА – біт парності, його наявність і тип.

N – None (Відсутній), без біта парності.

E – Even (Парний). Перевірка даних на парність. Перед стоп-бітом у кадр додається біт: 0, якщо у кадрі була непарна кількість одиниць, 1 – якщо парна.

O – Odd (Непарний). Перевірка даних на непарність. Перед стоп-бітом у кадр додається біт: 1, якщо у кадрі була непарна кількість одиниць, 0 – якщо парна.

ЦИФРА – тривалість стоп-біта 1, 1.5, 2.

Таким чином Serial Monitor у Arduino IDE працює з наступними стандартними налаштуваннями: **8-N-1**. Тобто вісім інформаційних біт, біт парності відсутній, стоп-біт має тривалість 1. Virtual Terminal у Proteus за замовчуванням має аналогічні налаштування.

Розглянемо конкретний приклад. Дана наступна часова діаграма (див. рисунок 7.3), необхідно визначити переданий у ASCII кодуванні рядок, якщо відомо, що тип передачі це 8-N-1, швидкість роботи протоколу (baud rate) – 9600, а біти йшли один за одним.

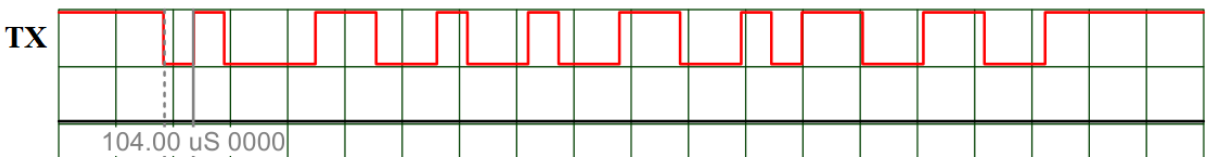


Рисунок 7.3 - Сигнал за протоколом UART

Відмітимо часові інтервали по 104 мкс .

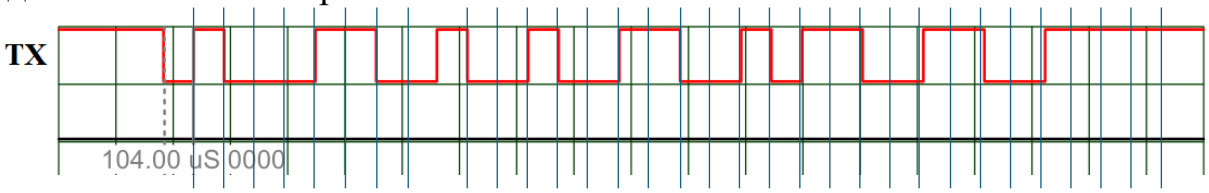


Рисунок 7.4 - Відокремлені біти сигналу UART

Підпишемо кожен біт:

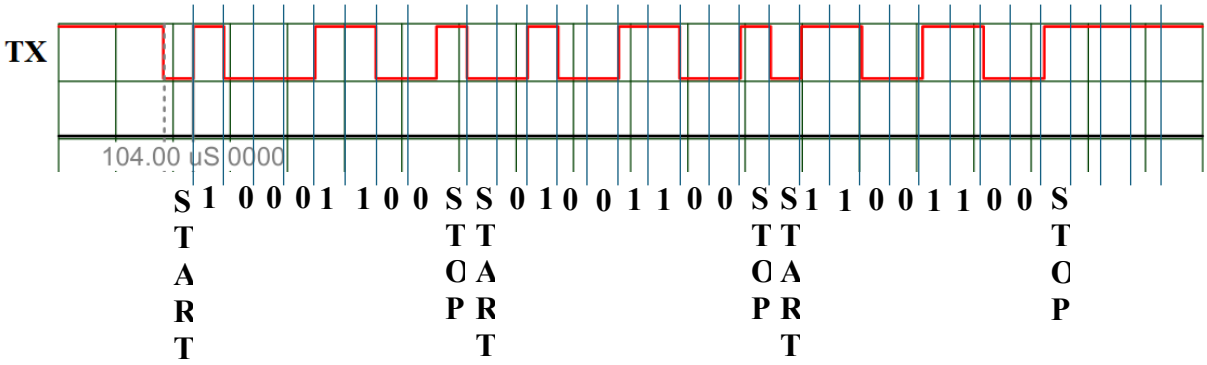


Рисунок 7.5 - Аналізуємо кожний біт сигналу UART

Запишемо біти, починаючи з молодшого:

00110001 = 0x31 = '1',

00110010 = 0x32 = '2'

00110011 = 0x33 = '3'

Отримаємо, що було передано наступний рядок: "123"

Розглянемо тепер використання інтерфейсу UART у мікроконтролерах.

У мікроконтролері ATmega328 на апаратному рівні підтримується один інтерфейс UART (n=0), а ATmega2560 – чотири (n=0,1,2,3). У таблиці 7.1 наведено піни, на які виведено інтерфейс UART у цих мікроконтролерах.

Таблица 7.1. – Интерфейс UART в мікроконтролерах ATmega328/2560.

№ UART	RX	TX
ATmega328		
UART0	RX0=PD0=D0	TX0=PD1=D1
ATmega2560		
UART0	RXD0=PE0=D0	TXD0=PE1=D1
UART1	RXD1=PD2=D19	TXD1=PD3=D18
UART2	RXD2=PH0=D17	TXD2=PH1=D16
UART3	RXD3=PJ0=D15	TXD3=PJ1=D14

Для використання апаратного інтерфейсу UART необхідно задіяти наступні регістри управління та стану (**USART Control and Status Register**):

UCSRnA, **UCSRnB, UCSRnC** и **UBRRn**.

А також регістрах даних:

UDRn.

На наступних рисунках зображено структуру відповідних регістрів.

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Рисунок 7.6 - Регістр UCSRnA.

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 7.7 - Регістр UCSRnB.

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

Рисунок 7.8 - Регістр UCSRnC.

15	14	13	12	11	10	9	8	
–	–	–	–	UBRRn[11:8]				UBRRnH
UBRRn[7:0]								UBRRnL
7	6	5	4	3	2	1	0	

Рисунок 7.9 Регістр UBRRn.

7	6	5	4	3	2	1	0	
RXB[7:0]								UDRn (Read)
TXB[7:0]								UDRn (Write)
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Рисунок 7.10 Регістр UDRn.

Розглянемо основні біти та їх призначення.

Регістр **UCSRnA**.

- ✓ RXCn (USART Receive Complete).

Цей біт прапора можна лише читати. Встановлюється, коли у приймальному буфері є непрочитані дані, і скидається, коли приймальний буфер порожній.

- ✓ TXCn (USART Transmit Complete).

Встановлюється, коли всі дані з буфера передачі (UDRn) пішли і зараз у ньому немає нових даних.

- ✓ U2Xn (Double the USART Transmission Speed).

Встановлення цього біта призводить до подвійної швидкості роботи USART в асинхронному режимі. Потрібно скинути цей біт у нуль під час використання синхронного режиму.

- ✓ UDREn (USART Data Register Empty).

Прапор UDREn вказує, готовий чи ні буфер передачі (UDRn) до прийому нових даних. Якщо UDREn дорівнює одиниці, буфер порожній і, отже, готовий до запису.

Регістр **UCSRnB**.

- ✓ RXCIEn (RX Complete Interrupt Enable n).

Запис цього біта в одиницю дозволяє переривання за вектором USART_RX_vect (див. таблицю 7.6 для ATmega328) по завершенню прийому, тобто коли буде автоматично виставлений біт RXC0 регістру **UCSR0A**. Для мікроконтролера ATmega2560 (див. таблицю 7.7) є чотири вектори переривань USARTn_RX_vect, де n=0,1,2,3. Глобальні переривання повинні бути дозволені.

- ✓ TXCIEn (TX Complete Interrupt Enable n).

Запис цього біта в одиницю дозволяє переривання за вектором USART_TX_vect (див. таблицю 7.6 для ATmega328) по завершенню передачі, тобто коли буде автоматично виставлений біт TXCn регістру **UCSR0A**. Для мікроконтролера

ATmega2560 (див. таблицю 7.7) є чотири вектори переривань USARTn_TX_vect, де n=0,1,2,3. Глобальні переривання повинні бути дозволені.

- ✓ UDRIEn (USART Data Register Empty Interrupt Enable n).

При встановленні цього біта в одиницю дозволяється переривання по прапору UDREn регістру **UCSRnA**, який встановлюється, якщо буфер порожній і, отже, в нього дозволено запис. Вектор переривання для ATmega328 згідно з таблицею 7.6 є USART_UDRE_vect. ATmega2560 згідно з таблицею 7.7 має чотири вектори переривання USARTn_UDRE_vect, де n=0,1,2,3. Глобальні переривання також повинні бути дозволені.

- ✓ RXENn (Receiver Enable n).

Установка цього біта в одиницю запускає USART на прийом, причому звичайний режим GPIO на відповідному піні працювати не буде.

- ✓ TXENn (Transmitter Enable n).

Установка цього біта в одиницю запускає USART на передачу, причому звичайний режим GPIO на відповідному піні працювати не буде. Скидання біта TXENn в нуль набуде чинності тільки після того, як всі поточні та очікувані передачі будуть завершені.

- ✓ UCSZn2 (Character Size n).

Біт UCSZn2 у поєднанні з бітами UCSZn1 і UCSZn0 регістру **UCSRnC** встановлює кількість біт, що передаються по UART.

Регистр **UCSRnC**.

- ✓ Біти UMSELn1 та UMSELn0 (USART Mode Select) відповідають за режим роботи згідно з таблицею 7.2.

Таблиця 7.2. – Синхронно асинхронний режим роботи.

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART *режим за замовчуванням*
0	1	Synchronous USART

UMSELn1	UMSELn0	Mode
1	0	(Reserved)
1	1	Master SPI (MSPIM)

- ✓ Бітами UPMn1 та UPMn0 (USART Parity Mode) налаштовується режим перевірки парності/непарності (див. таблицю 7.3).

Таблиця 7.3. – Режим перевірки парності/непарності.

UPMn1	UPMn0	Parity Mode
0	0	Disabled *режим за замовчуванням*
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- ✓ Біт USBSn вказує кількість стопових бітів: якщо USBSn=0, то стоповий біт 1, якщо USBSn=1, то 2.
- ✓ Біти UCSZn2 (реєстр UCSRnB), UCSZn1, UCSZn0 визначають кількість інформаційних біт під час прийому/передачі.

Таблиця 7.4. – Кількість інформаційних біт під час прийому/передачі.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit *режим за замовчуванням*
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Регістр **UDRn**.

Це UART Data Register. Тобто це регістр, в який потрібно помістити байт для відправки. Також з цього регістра отримують надісланий іншим пристроєм байт даних. Насправді це два абсолютно різних регістри (див. рисунок 7.10), але

вони мають одну адресу. При цьому коли ми записуємо байт, він потрапляє в TXB, а коли читаємо, то з RXB.

Регістр **UBRRn.**

Цей регістр визначає швидкість роботи за протоколом UART. У технічній документації наводяться такі формули у разі асинхронної роботи інтерфейсу:

$$UBRRn = \frac{fosc}{16 \cdot BAUD} - 1, \quad (7.1)$$

$$UBRRn = \frac{fosc}{8 \cdot BAUD} - 1, \quad (7.2)$$

де $UBRRn$ – значення однойменного регістру, f_{OSC} – робоча частота мікроконтролера, $BAUD$ – швидкість роботи UART.

Формула (7.1) справедлива, якщо біт U2Xn = 0, а формула (7.2) - якщо біт U2Xn = 1. Біт U2Xn це перший біт регістру UCSRnA (див. рисунок 7.6).

Однак найпростіше скористатися таблицею, яка також наведена у технічній документації.

Таблиця 7.5. – Зв'язок між значенням регістру UBRRn та швидкістю роботи.

	f_{osc} = 16.0000 MHz			
	U2Xn = 0		U2Xn = 1	
Baud Rate (bps)	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1 Mbps		2 Mbps	

З цієї таблиці видно, що при частоті мікроконтролера 16 МГц і бажаної швидкості роботи 9600 бод значення регістра UBRRn дорівнює 103 в стандартному режимі, та 207 – при подвоєному.

Зазначимо ще один важливий момент. Структуру регістру UBRRn наведено рисунку 7.9. З якого видно, що він складається з двох восьми-бітових регістрів UBRRnH та UBRRnL. Причому використовуються лише молодші чотири біти старшого регістру UBRRnH. Таким чином, при записі значення UBRRn ми повинні записати спочатку старші біти в регістр UBRRnH, а потім молодші в UBRRnL.

Для вивчення інтерфейсу UART за допомогою Proteus необхідно зібрати схему, зображену на рисунку 7.11 якщо ми працюємо з ATmega328, або схему яку зображено на рисунку 7.12, якщо використовується ATmega2560.

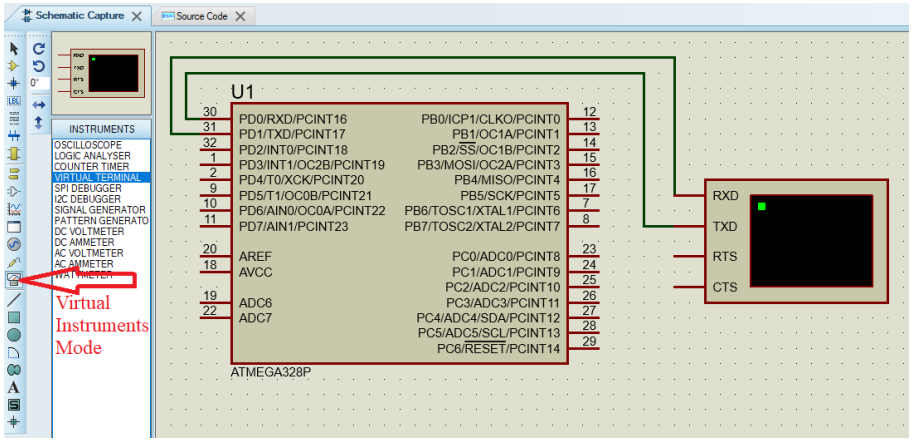


Рисунок 7.11 - Підключення Virtual Terminal до ATmega 328

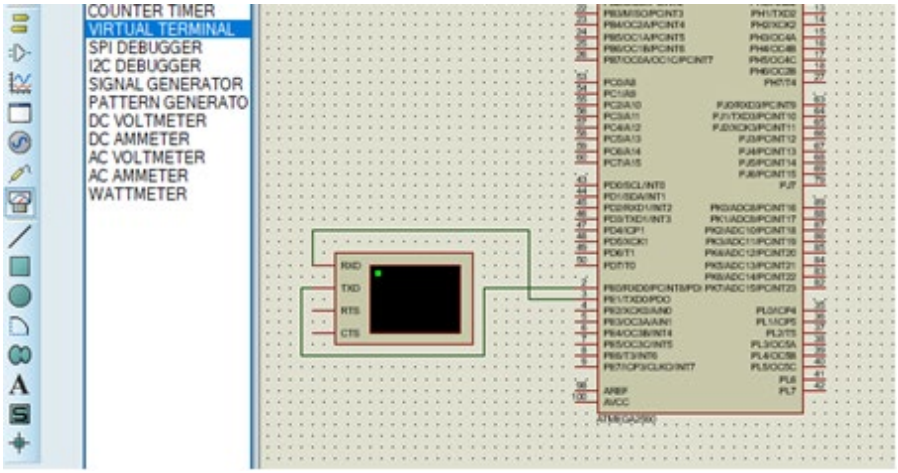


Рисунок 7.12 - Підключення Virtual Terminal до ATmega 2560

Зверніть увагу, що підключення пристрою з UART перехресне, тобто пін RX з'єднується з TX, а TX з RX. Оскільки ми хочемо, щоб лабораторні роботи можна було виконувати або в симуляторі, або на реальному устаткуванні, то для сумісності будемо використовувати лише

UART0 на ATmega2560, так само як і платформа Arduino MEGA яка з'єднана з Serial Monitor за допомогою UART0.

Також необхідно переконатися, що Virtual Monitor налаштований аналогічно Serial Monitor в Arduino IDE, тобто. 8-N-1. Для цього двічі клацаємо лівою кнопкою миші по Virtual Monitor і налаштовуємо обладнання, як зазначено на рисунку 7.13.

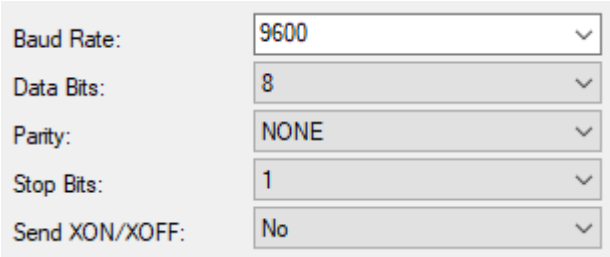


Рисунок 7.13 - Налаштування Virtual Monitor у Proteus.

Наведемо програму **Prog7.1**, яка виведе в монітор порту фразу HELLO WORLD.

Prog7.1

```
#define F_CPU 16000000UL //16MHz
#include <avr/io.h>
#include <util/delay.h>

// функція ініціалізації UART
void UART_init(){
  // налаштування швидкості передачі 9600 бод
  UBRR0H = 0;
  UBRR0L = 103; // (16000000/(9600*16))-1=103 Table 7.5

  //Налаштування
  //UCSR0A  U2Xn=0 одинарна швидкість передачі

  //UCSR0B
  //RXCIE0=TXCIE0=UDRIE0=0   не використовуємо переривання
  //RXEN0=0                  не використовуємо прийом даних
  UCSR0B |= (1 << TXEN0);    // дозволити передачу
  //UCSZ02=0                 задає кількість біт, що передаються

  //UCSR0C
  //UMSEL01=UMSEL00=0        асинхронний режим
  //UPM01=UPM00=0            контроль парності відсутній
  //USBS0=0                  стоповий біт один
  //UCSZ01=UCSZ00=1          з урахуванням UCSZ02 передаємо 8 біт
}

// Функція передачі одного символу
void UART_transmit_char( uint8_t c ){
   // чекаємо поки буфер передачі не звільниться
   while( !(UCSR0A & (1 << UDRE0)) );
   UDR0 = c;
}

// Функція передачі рядка
void UART_transmit_str(char *str){
    uint8_t c;
     while((c =*str++)!= 0){// передаємо символи послідовно
        UART_transmit_char(c);
```

```
39        }
40   }
41   int main(){
42     UART_init(); // ініціалізуємо UART
43       while(1){
44         UART_transmit_str("Hello World\r\n");
45         _delay_ms(300);
46        }
47   }
```

Завантаживши програму і ввімкнувши Serial Monitor (не забудьте налаштувати швидкість його роботи 9600 бод), після цього побачимо фразу **`Hello World`**.

Програма для виведення числа за інтерфейсом UART

Prog7.2

```
1    #define F_CPU 16000000UL //16MHz
2    #include <avr/io.h>
3    // функція ініціалізації UART
4    void UART_init(){
5      // налаштування швидкості передачі 9600 бод
6      UBRR0H = 0;
7      UBRR0L = 103; // (16000000/(9600*16))-1=103 Table 7.5
8      //Налаштування
9      //UCSR0A  U2Xn=0 одинарна швидкість передачі
10
11     //UCSR0B
12     //RXCIE0=TXCIE0=UDRIE0=0   не використовуємо переривання
13     //RXEN0=0                  не використовуємо прийом даних
14     UCSR0B |= (1 << TXEN0);    // дозволити передачу
15     //UCSZ02=0                 задає кількість біт, що передаються
16
17     //UCSR0C
18     //UMSEL01=UMSEL00=0        асинхронний режим
19     //UPM01=UPM00=0            контроль парності відсутній
20     //USBS0=0                  стоповий біт один
21     //UCSZ01=UCSZ00=1          з урахуванням UCSZ02 передаємо 8 біт
22   }
23   // Функція передачі однієї цифри d
24   void UART_transmit_digit( uint8_t d ){
25      // чекаємо поки буфер передачі не звільниться
26      while( !(UCSR0A & (1 << UDRE0)) );
27      UDR0 = d+0x30;           // формуємо код ASCII заданої цифри
28   }
29   // Функція передачі цілого беззнакового числа D
30   void UART_transmit_uint( uint32_t D ){
31     uint8_t M[10], i=0;// Масив зберігає число-у кожній комірці цифра
32       do {
33         M[i]=D%10;            //записуємо цифру молодшого розряду
34         D/=10;                // відкидаємо від числа молодший розряд
35         i++;                  // збільшуємо лічильник цифр у числі на 1
36       }
37       while (D>0);            // якщо D=0, усі цифри числа перебрали
38       for (;i>0;i--){         // починаємо цикл з і рівною кількістю цифр
39         UART_transmit_digit(M[i-1]);//передаємо цифри зі старшого розряду
40       }
41   }
42   int main(){
43     UART_init();                  // ініціалізуємо UART
44     UART_transmit_uint(12345);    // передаємо ціле не від'ємне число
45       while(1){}
46   }
```

Напишемо тепер програму, яка керуватиме світлодіодом на піні D13 (ATmega328) за допомогою UART. Коли посилаємо 1, стан світлодіоду буде змінюватися на протилежний.

Prog7.3

```
#define F_CPU 16000000UL //16MHz
#include <avr/io.h>

// функція ініціалізації UART
void UART_init(){
  // налаштування швидкості передачі 9600 бод
  UBRR0H = 0;
  UBRR0L = 103; //(16000000/(9600*16))-1=103 для 16 МГц швидкость 9600

  // Налаштування
  //UCSR0A  U2Xn=0 одинарна швидкість передачі

  //UCSR0B
  //RXCIE0=TXCIE0=UDRIE0=0   не використовуємо переривання
  //TXEN0=0                  не використовуємо передачу даних
  UCSR0B |= (1 << RXEN0);   // дозволити прийом
  //UCSZ02=0                 задає кількість біт, що передаються

  //UCSR0C
  //UMSEL01=UMSEL00=0        асинхронний режим
  //UPM01=UPM00=0            контроль парності відсутній
  //USBS0=0                  стоповий біт один
  //UCSZ01=UCSZ00=1          з урахуванням UCSZ02 передаємо 8 біт
}

//Функція для прийому символа
uint8_t UART_receive(){
   // чекаємо прийом байту
    while((UCSR0A&(1 << RXC0))==0);
   // зчитуємо прийнятий байт
   return UDR0;
}
uint8_t ControlByte;           // змінна для керування світлодіодом
int main(){
  DDRB |= (1<<DDB5);
  PORTB &=~(1<<PORTB5);
  UART_init();                 // ініціалізуємо UART
    while(1){
      ControlByte = UART_receive();
      if (ControlByte=='1'){
        PORTB ^=(1<<PORTB5);   // інверсія стану світлодіода
       }
    }
return 0;
}
```

Якщо ми працюємо з Virtual Monitor, то для того щоб можна було бачити символи, що посилаються через UART потрібно включити Echo Typed Characters (див. рисунок 7.14) для цього потрібно правою кнопкою миші клацнути в робочому полі Virtual Monitor і вибрати відповідний пункт. Після цього символи будуть відображатися у вікні.

Зауважимо, що на відміну від Serial Monitor (Arduino IDE) символи, що відображаються, в Virtual Monitor відразу посилаються по UART інтерфейсу, в Serial Monitor потрібно натиснути ENTER.

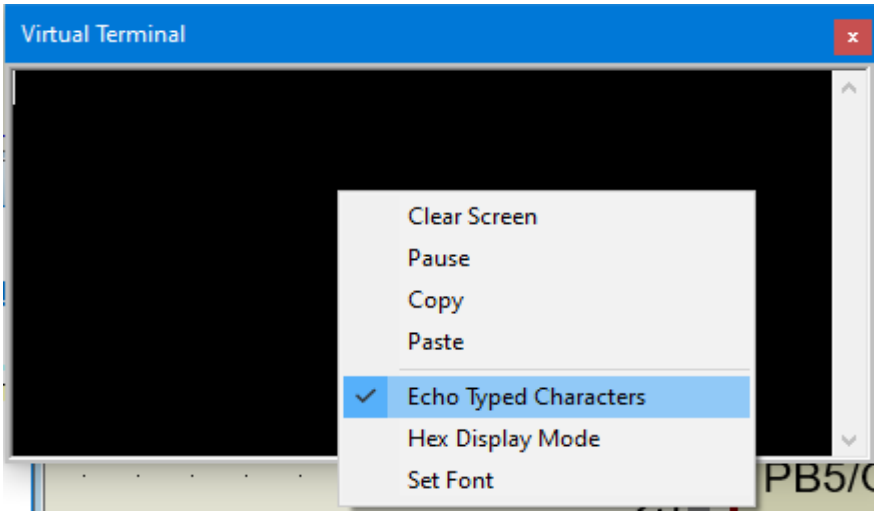


Рисунок 7.14 - Налаштування Virtual Monitor для передачі даних UART.

Найбільший недолік програми **Prog7.3** полягає в тому, що в головному нескінченному циклі (рядки 39-43) мікроконтролер змушений чекати на прихід байту по UART і при цьому нічим іншим він займатися не може. Щоб уникнути цього, потрібно використовувати переривання.

Таблиця 7.6 – USART вектори переривань для ATmega328

Номер вектора	Адреса	Джерело	Опис переривання
19	0x0024	USART, RX	USART прием закінчений
20	0x0026	USART, UDRE	Регістр даних USART порожній
21	0x0028	USART, TX	USART передача закінчена

Таблиця 7.7 – USART вектори переривань для ATmega2560

Vector No.	Program Address	Source	Interrupt Definition
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
52	$0066	USART2 RX	USART2 Rx Complete
53	$0068	USART2 UDRE	USART2 Data Register Empty
54	$006A	USART2 TX	USART2 Tx Complete
55	$006C	USART3 RX	USART3 Rx Complete
56	$006E	USART3 UDRE	USART3 Data Register Empty
57	$0070	USART3 TX	USART3 Tx Complete

Напишемо наступну програму

Prog7.4

```
1    #define F_CPU 16000000UL //16MHz
2    #include <avr/io.h>
3    #include <avr/interrupt.h>  // Бібліотека імен переривань
4
5    //функція ініціалізації UART
6    void UART_init(){
7      // налаштування швидкості передачі 9600 бод
8      UBRR0H = 0;
9      UBRR0L = 103; //(16000000/(9600*16))-1=103 для 16 МГц швидкость 9600
10
11     // Налаштування
12     //UCSR0A  U2Xn=0 одинарна швидкість передачі
13
14     //UCSR0B
15     //TXCIE0=UDRIE0=0          не використовуємо переривання
16     UCSR0B |= (1 << RXCIE0);  //дозволяємо переривання після прийому
17     //TXEN0=0                  не використовуємо передачу даних
18     UCSR0B |= (1 << RXEN0);   // дозволити прийом
19     //UCSZ02=0               задає кількість біт, що передаються
20
21     //UCSR0C
22     //UMSEL01=UMSEL00=0      асинхронний режим
23     //UPM01=UPM00=0          контроль парності відсутній
24     //USBS0=0                стоповий біт один
25     //UCSZ01=UCSZ00=1        з урахуванням UCSZ02 передаємо 8 біт стр 191
26   }
27
28   int main(){
29     DDRB |= (1<<DDB5);
30     PORTB &=~(1<<PORTB5);
31     UART_init();               // ініціалізуємо UART
32     sei();
33       while(1){}
34   return 0;
35   }
36    ISR(USART_RX_vect){
37     uint8_t cb = UDR0;         //cb=control byte
38     if (cb=='1'){
39           PORTB ^=(1<<PORTB5);
40         }
41     if (cb=='H'){
42           PORTB |=(1<<PORTB5);
43         }
44     if (cb=='L'){
45           PORTB &=~(1<<PORTB5);
46         }
47    }
```

У програмі **Prog7.4** основна частина (рядок 33) порожня, всі дії відбуваються у перериваннях. Коли вводиться 1, стан світлодіоду на D13 інвертується, якщо вводимо H, то вмикаємо світлодіод, якщо L – вимикаємо.

Зазначимо, якщо цю програму треба переписати під ATmega2560, то потрібно пін PB5 замінити на PB7, а також вектор **`USART_RX_vect`** замінити на **`USART0_RX_vect`**.

Зараз розробимо програму **Prog7.5** яка буде отримувати по інтерфейсу UART число типу **`uint8_t`** та передавати це число в порівняльний регістр каналу A таймер-лічильника №0, який налаштовано на генерацію ШІМ сигналу. Таким чином ми зможемо з клавіатури встановлювати потрібну яскравість світлодіоду, якщо він буде підключено до відповідного піну.

Проект розробимо на ATmega2560, тому ШІМ сигнал будемо генерувати на піні `PB7=OC0A=D13`. Після того, як усі цифри потрібного числа будуть введені, необхідно ввести `q` – це дозволить програмі зрозуміти, що число введено і можна записувати ШІМ параметр. На жаль, Proteus не може змінювати яскравість світлодіоду при симуляції роботи програми, тому разом зі світлодіодом до вказаного піна ми підключимо осцилограф, який в реальному часі буде демонструвати згенерований ШИМ сигнал (див. рисунок 7.15).

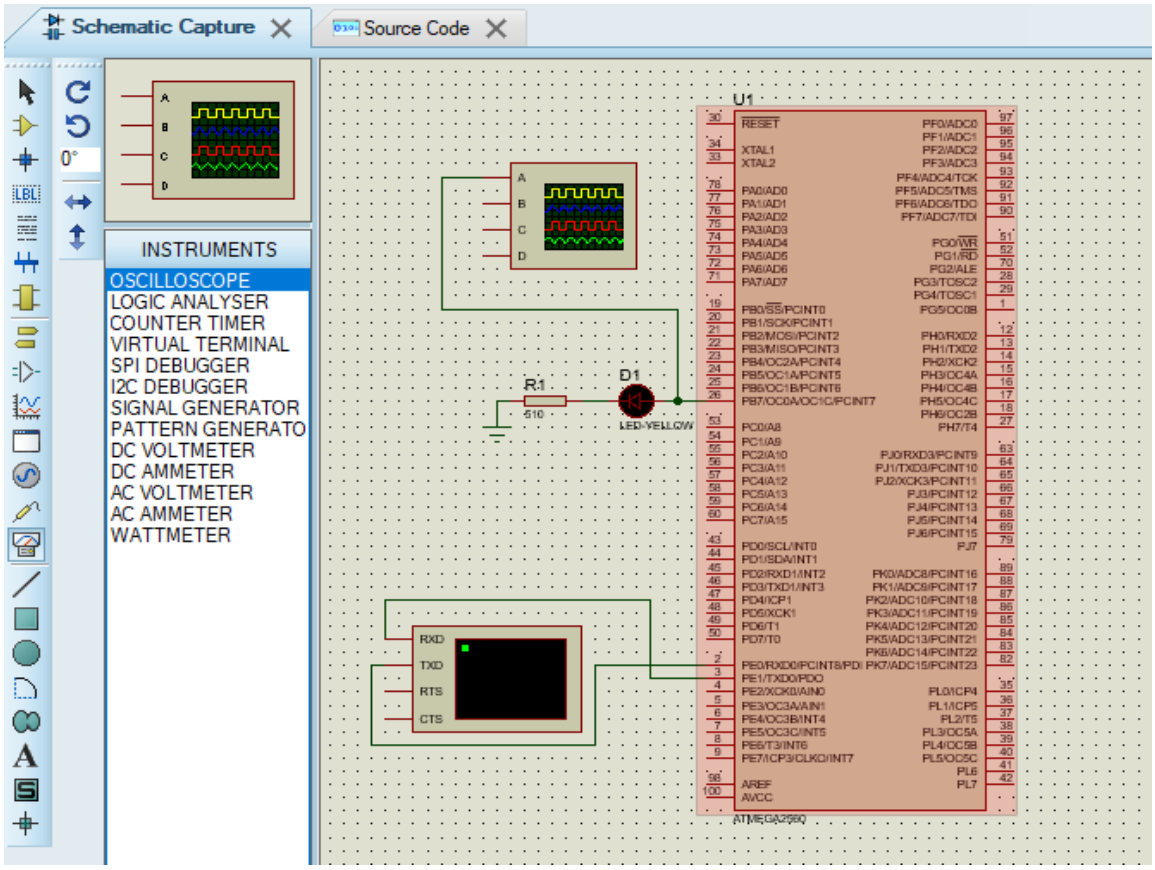


Рисунок 7.15 - Керування ШІМ сигналом за допомогою UART

Prog7.5

```
#define F_CPU 16000000UL    //16MHz
#include <avr/io.h>
#include <avr/interrupt.h>  // Бібліотека імен переривань
uint8_t bufRX[5];          // по розрядно містить ШІМ параметр
volatile uint8_t count=0;  // кількість розрядів ШІМ параметра

//функція ініціалізації UART
void UART_init(){
  // налаштування швидкості передачі 9600 бод
  UBRR0H = 0;
  UBRR0L = 103; //(16000000/(9600*16))-1=103 для 16 МГц швидкость 9600

  // Налаштування
  //UCSR0A  U2Xn=0 одинарна швидкість передачі

  //UCSR0B
  //TXCIE0=UDRIE0=0         не використовуємо переривання
  UCSR0B |= (1 << RXCIE0);  // дозволяємо переривання після прийому
  //TXEN0=0                  не використовуємо передачу даних
  UCSR0B |= (1 << RXEN0);   // дозволити прийом
  //UCSZ02=0               задає кількість біт, що передаються

  //UCSR0C
  //UMSEL01=UMSEL00=0      асинхронний режим
  //UPM01=UPM00=0          контроль парності відсутній
  //USBS0=0                стоповий біт один
  //UCSZ01=UCSZ00=1        з урахуванням UCSZ02 передаємо 8 біт стр 191
}
int main(){
  //налаштовуємо ШІМ on ATmega2560
  DDRB |= (1<<DDB7);                //PB7=OC0A=D13 як ВИХІД
  TCCR0A |= (1<<WGM00);             //режим PWM Phase Correct №1
  TCCR0B |= (1<<CS02)|(1<<CS00);    //дільник 1024
  TCCR0A |= (1<<COM0A1);            //неінвертований ШІМ

  UART_init();                      // ініціалізуємо UART
  sei();
   while(1){}                       //головна частина
}
ISR(USART0_RX_vect){
  uint8_t udr = UDR0;
  if (udr != 'q'){                // наприкінці числа посилаємо'q'=0x71
    bufRX [count++] = udr-0x30;   // переводимо код ASCII у цифру
  }
  else{             // збираємо отримане число із цифр (max PWM = 255)
    switch(count){
       case 3:
         OCR0A = bufRX[0]*100+bufRX[1]*10+bufRX[2];
         break;
       case 2:
         OCR0A = bufRX[0]*10+bufRX[1];
         break;
       case 1:
         OCR0A = bufRX[0];
         break;
    }
    count=0;
  }
}
```

Після завантаження програми пишемо в Virtual Terminal `10q` (Enter не натискаємо) і бачимо на осцилографі відповідний ШІМ сигнал (див. рисунок 7.16).

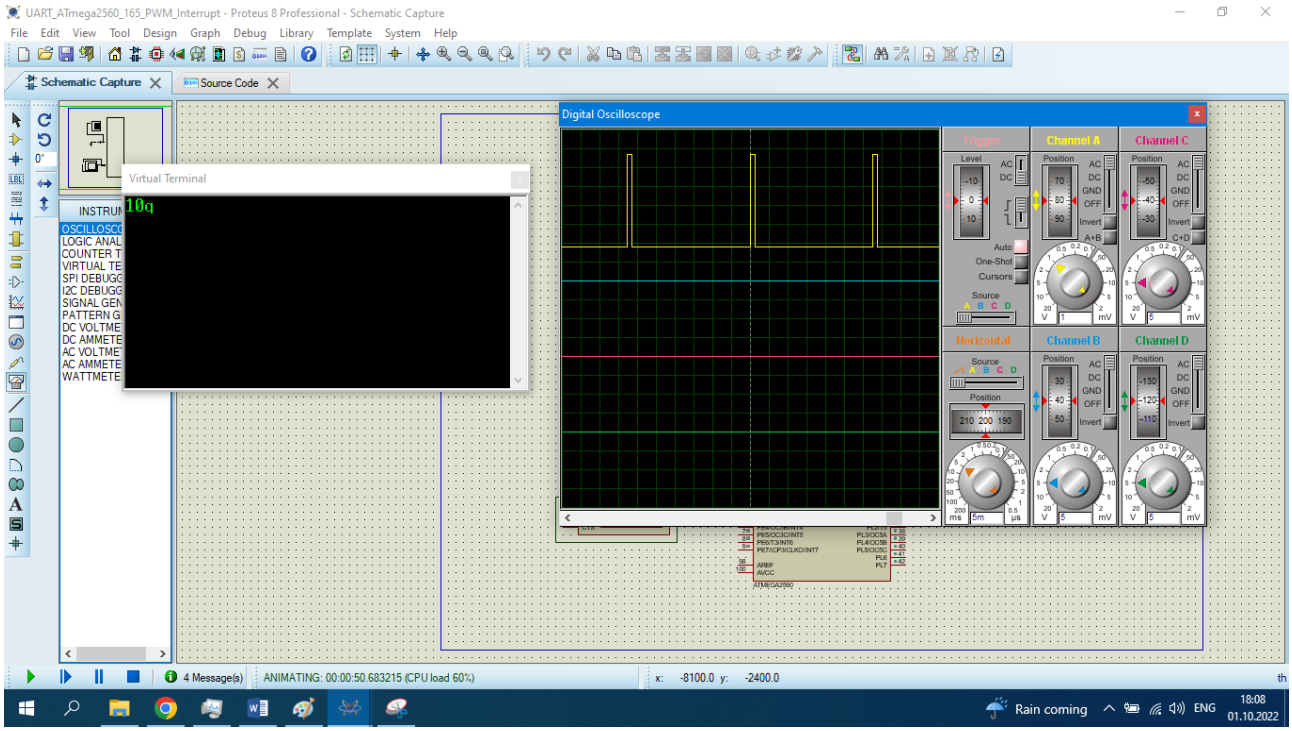


Рисунок 7.16 - Задаємо ШИМ параметр 10 за допомогою UART.

Не зупиняючи симуляцію, вводимо `50q` і бачимо відповідні зміни ШІМ сигналу (див. рисунок 7.17).

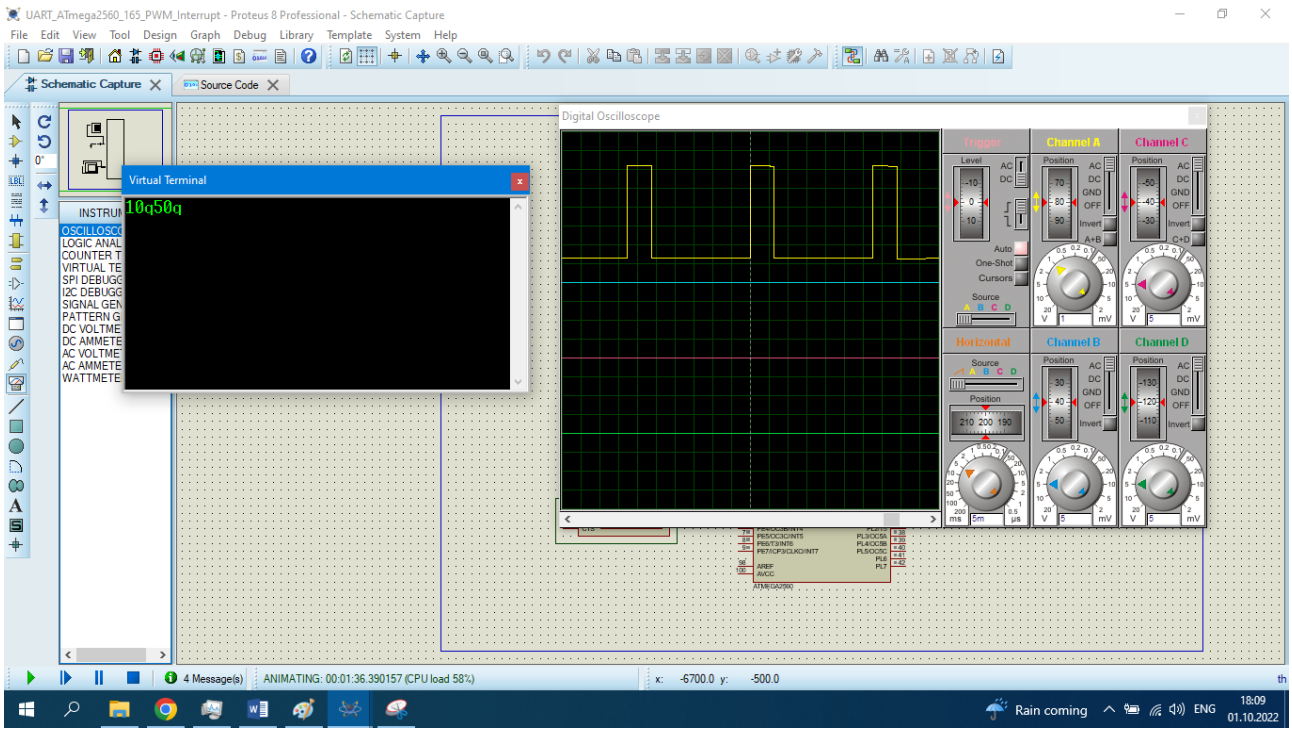


Рисунок 7.17 - Задаємо ШИМ параметр 50 за допомогою UART.

Аналогічно задавши ШІМ параметр 150 і 250 можна бачити зміни сигналу, які формуються на піні OC0A (див. рисунки 7.18 та 7.19).

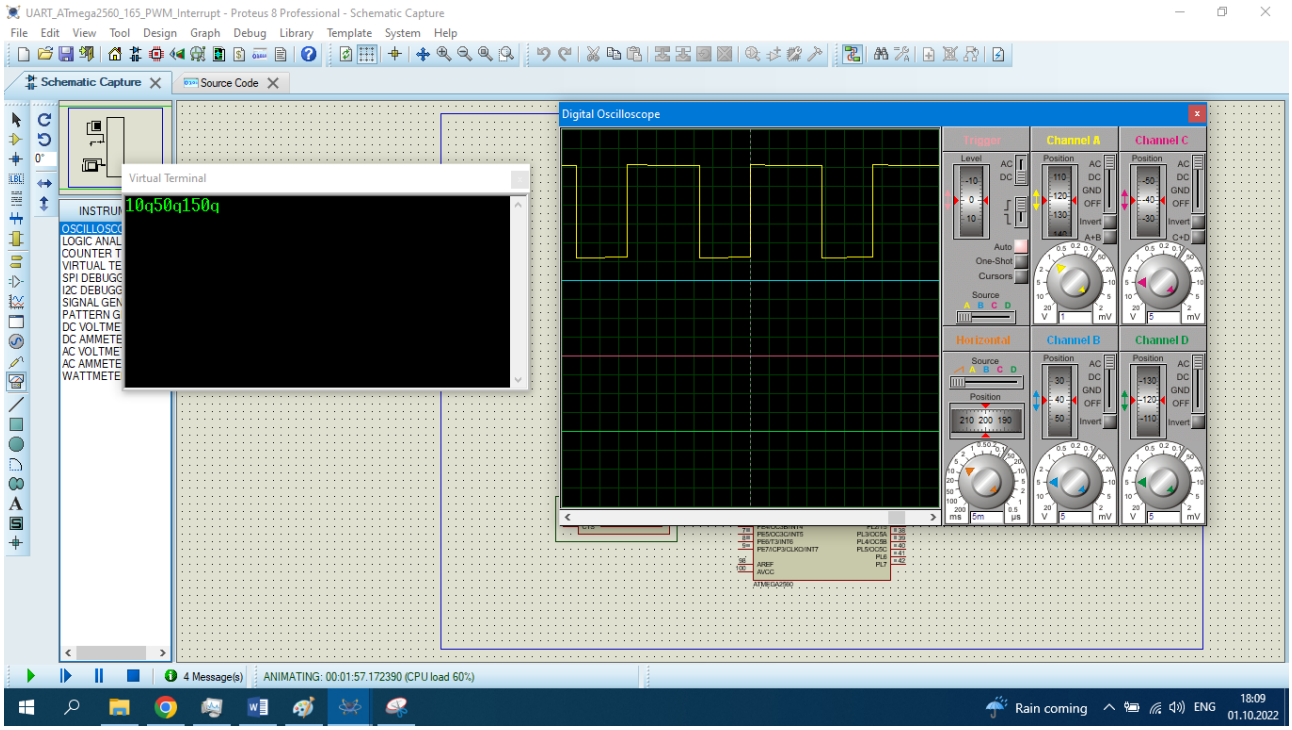


Рисунок 7.18 - Задаємо ШИМ параметр 150 за допомогою UART.

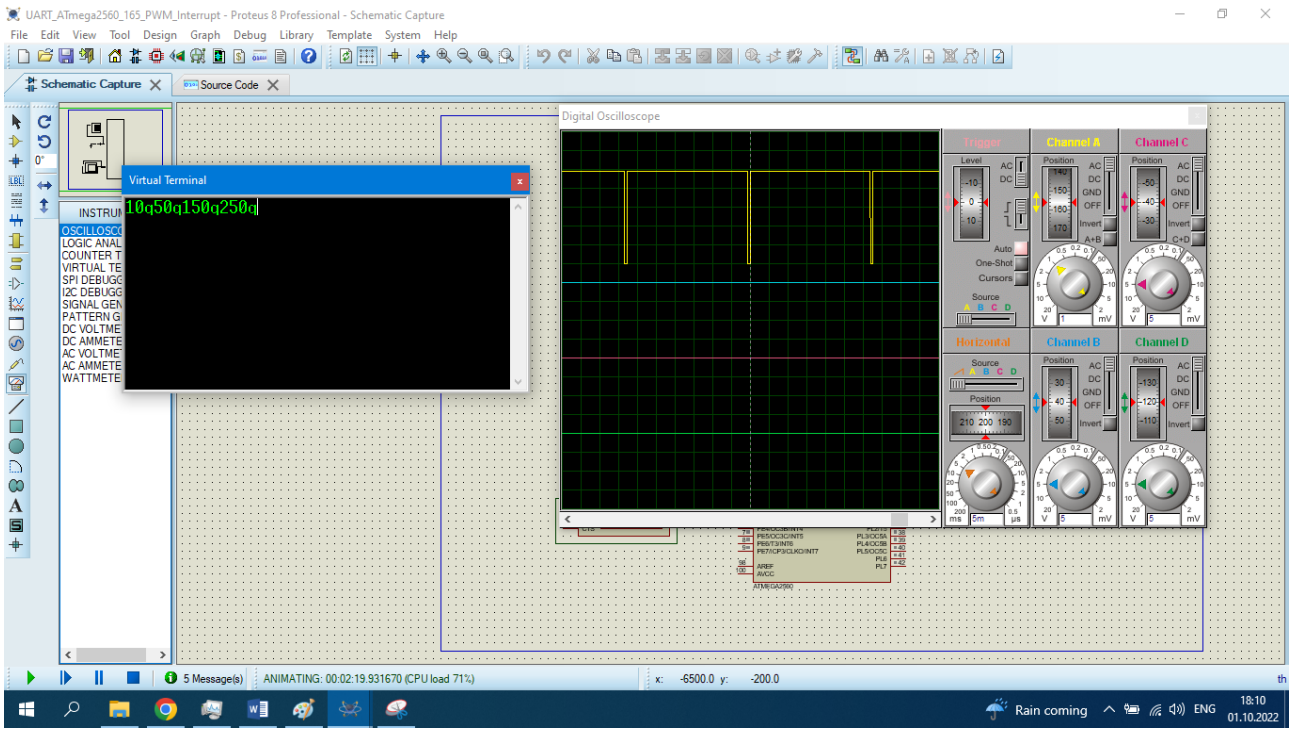


Рисунок 7.19 - Задаємо ШИМ параметр 250 за допомогою UART.

Хід роботи

Завдання 7.1.

Написати програму ЛУНА яка **не використовує** переривання. Тобто посилаємо один символ по UART та отримуємо у відповідь той самий символ. Згідно таблиці виберіть швидкість роботи UART інтерфейсу.

Таблиця 7.8. – Вхідні данні для завдання 7.1

Варіант	BAUD	U2X0
1	2400	1
2	4800	1
3	19200	1
4	38400	1
5	57600	1
6	1000000	1
7	115200	1
8	250000	1
9	2400	0
10	4800	0
11	19200	0
12	38400	0
13	57600	0
14	1000000	0
15	2400	1
16	250000	0
17	500000	0
18	9600	1
19	115200	1
20	500000	1

Примітка: у вікні властивостей Virtual Terminal в полі Baud Rate можливо просто ввести необхідне значення швидкості.

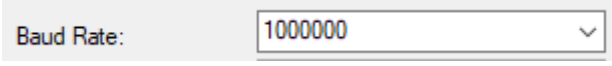


Рисунок 7.20 – Встановлення швидкості роботи Virtual Terminal

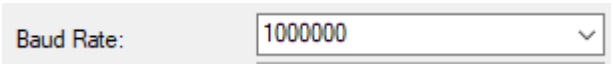
Завдання 7.2.

Написати програму ЛУНА яка **використовує** переривання. Тобто посилаємо один символ по UART та отримуємо у відповідь той самий символ. Згідно таблиці виберіть швидкість роботи UART інтерфейсу.

Таблиця 7.9. – Вхідні данні для завдання 7.2

Варіант	BAUD	U2X0
1	4800	1
2	19200	1
3	38400	1
4	57600	1
5	1000000	1
6	115200	1
7	250000	1
8	2400	0
9	4800	0
10	19200	0
11	38400	0
12	57600	0
13	1000000	0
14	2400	1
15	250000	0
16	500000	0
17	9600	1
18	115200	1
19	500000	1
20	2400	1

Примітка: у вікні властивостей Virtual Terminal в полі Baud Rate можливо просто ввести необхідне значення швидкості.



Завдання 7.3.

Написати програму яка буде отримувати по UART ціле число x типу int16_t, тобто ціле число зі знаком, та повертати у термінал результат розрахунку відповідної функції $f(x)$ типу int32_t. Надати відповідні скріншоти в яких є вивід у терміналі позитивних і негативних чисел.

Таблиця 7.10. – Вхідні данні для завдання 7.3

Варіант	$f(x)$	BAUD	U2X0
1	$x^2-10x+5$	4800	1
2	$2x^2-5x-3$	19200	1
3	$3x^2-8x-23$	38400	1
4	$4x^2-80x+3$	57600	1
5	$5x^2-20x-70$	1000000	1
6	$-x^2+2x+44$	115200	1
7	$-2x^2+x+30$	250000	1
8	$-3x^2+5x+25$	2400	0
9	$-4x^2+3x+57$	4800	0
10	$-5x^2+2x+28$	19200	0
11	$x^2-10x+5$	38400	0
12	$2x^2-5x-3$	57600	0
13	$3x^2-8x-23$	1000000	0
14	$4x^2-80x+3$	2400	1
15	$5x^2-20x-70$	250000	0
16	$-x^2+2x+44$	500000	0
17	$-2x^2+x+30$	9600	1
18	$-3x^2+5x+25$	115200	1
19	$-4x^2+3x+57$	500000	1
20	$-5x^2+2x+28$	2400	1

Лабораторна робота №8

АЦП

Мета: навчиться використовувати вбудований АЦП модуль мікроконтролерів ATmega328 та ATmega2560 у різних режимах роботи.

Теоретичні відомості

У всіх мікроконтролерах серії ATmega розробники вбудували модуль аналого-цифрового перетворення. У цій лабораторній роботі ми розглянемо основні загальні можливості модулів АЦП мікроконтролерів ATmega328 і ATmega2560. Модуль АЦП має роздільну здатність 10 біт, це означає, що максимальний код, який можна отримати після перетворення аналогового сигналу дорівнює $1111111111_2 = 3FF_{16} = 1023_{10}$ який буде відповідати напрузі опорного джерела за вирахуванням ціни одного розряду. Тому розробники дають наступну формулу, що зв'язує *ADC* - код, який отримується на АЦП та вхідну напругу V_{in}

$$ADC = \frac{V_{in} \cdot 1024}{V_{ref}}, \tag{8.1}$$

де V_{ref} - опорна напруга. Іншими словами, при опорній напрузі яка дорівнює 5 вольтам, максимальна напруга, яку можна виміряти є

$$V_{in} = \frac{ADC \cdot V_{ref}}{1024} = \frac{1023 \cdot 5}{1024} = 4{,}995117 \text{ В}. \tag{8.2}$$

Для налаштування та роботи з АЦП ми будемо використовувати наступні регістри: `ADCSRA`, `ADCSRB`, `ADMUX`, `ADC`.

Почнемо з розгляду регістру `ADCSRA`.

У мікроконтролерах ATmega328 і ATmega2560 цей регістр однаковий. Згідно з описом розробника, його назва розшифровується як регістр статусу та управління АЦП (ADC Control and Status Register A) і має такий вигляд

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Рисунок 8.1 - Регістр `ADCSRA`

Опишемо коротко біти цього регістру.

Bit 7 – ADEN: ADC Enable

Установка цього біта в одиницю включає АЦП. Скидання його в нуль відключає АЦП. Якщо скинути біт під час аналого-цифрового перетворення, то поточне перетворення буде припинено.

Bit 6 – ADSC: ADC Start Conversion.

У режимі одиночного перетворення потрібно щоразу встановлювати цей біт в одиницю для запуску перетворення. У режимі Free Running встановлення цього біта в одиницю запускає перше перетворення. Біт ADSC буде зчитуватися як одиниця, доки виконується перетворення. Коли перетворення завершиться, цей біт буде скинуто в нуль. Запис нуля у цей біт не призводить до жодного ефекту.

Bit 5 – ADATE: ADC Auto Trigger Enable

Коли цей біт встановлюється в одиницю, включається автоматичний запуск АЦП. АЦП почне перетворення позитивним фронтом обраного тригерного сигналу. Джерело запуску вибирається встановленням бітів вибору запуску: ADTS2:0 у регістрі ADCSRB.

Bit 4 – ADIF: ADC Interrupt Flag

Цей біт встановлюється апаратно в одиницю, коли перетворення АЦП завершується та регістри даних оновлюються. Якщо переривання було дозволено бітом ADIE цього регістра і бітом I в регістрі SREG, відбувається запуск відповідного вектору обробки переривання і біт ADIF скидається в нуль апаратно. Можна очистити біт ADIF програмними засобами шляхом запису логічної *одиниці* до нього.

Bit 3 – ADIE: ADC Interrupt Enable

Коли цей біт встановлюється в одиницю і біт I в SREG встановлено, дозволяється запуск переривання після завершення перетворення АЦП.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits

Ці біти задають коефіцієнт поділу між системною тактовою частотою та вхідною тактовою частотою АЦП. Коефіцієнт поділу визначається таблицею 8.1 (див. Table 23-5 для ATmega328 або Table 130 для ATmega2560).

Таблиця 8.1. – Встановлення дільника в АЦП (Prescaler Selections)

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регістр статусу та управління АЦП `ADCSRB`.

У мікроконтролерах ATmega328 та ATmega2560 цей регістр відрізняється одним третім бітом. На рисунку 8.2 він позначений зеленим кольором. У мікроконтролері ATmega328 цей біт відсутній.

7	6	5	4	3	2	1	0	
–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0	ADCSRB
R	R/W	R	R	R	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Рисунок 8.2 - Регістр `ADCSRB`

Bit 6 – ACME: Analog Comparator Multiplexer Enable

Цей біт необхідний для вибору негативного входу під час використання аналогового компаратора. У режимі АЦП ми його не використовуємо.

Bit 3 – **MUX5**

Використовується для вибору додаткових режимів роботи мікроконтролера ATmega2560 (див. Table 129 документації розробника [2]).

Bit 2:0 – ADTS2:0: ADC Auto Trigger Source.

За допомогою цих біт визначається джерело сигналу (якщо включений біт ADATE регістра ADCSRA), яке буде запускати аналого-цифрове перетворення.

Таблица 8.2. – ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Джерело сигналу
0	0	0	безперервне перетворення
0	0	1	аналоговий компаратор
0	1	0	зовнішнє переривання INT0
0	1	1	за збігом таймера/лічильника T0 на каналі A

1	0	0	по переповненню таймера/лічильника T0
1	0	1	за збігом таймера/лічильника T1 на канале B
1	1	0	по переповненню таймера/лічильника T1
1	1	1	по захопленню таймера/лічильника T1

Регістр статусу та управління АЦП ADMUX.

У мікроконтролерах ATmega328 та ATmega2560 цей регістр відрізняється одним четвертим бітом. На рисунку 8.3 він позначений зеленим кольором. У мікроконтролері ATmega328 цей біт відсутній.

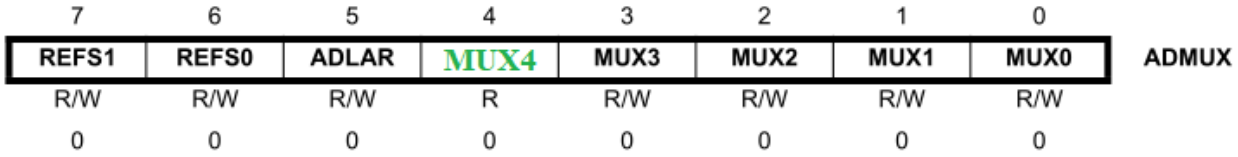


Рисунок 8.3 - Регістр ADMUX

Bit 7:6 – REFS1:0: Reference Selection Bits

Це бітове поле визначає джерело опорної напруги (ДОН). У таблиці 8.3 синім кольором виділено відмінності для ATmega2560.

Таблиця 8.3. – Voltage Reference Selections for ADC

REFS1	REFS2	Тип ДОН
0	0	зовнішній ДОН, внутрішній ДОН вимкнено
0	1	напруга живлення AVcc
1	0	стан зарезервовано (ATmega328) внутрішній ДОН 1.1В (ATmega2560)
1	1	внутрішній ДОН 1.1В (ATmega328) внутрішній ДОН 2.56В (ATmega2560)

Bit 5 – ADLAR: ADC Left Adjust Result

За допомогою цього біту вибирають тип представлення результату перетворення. При записі в цей біт логічної 1 результат АЦП в регістрах ADCL і ADCH буде записано в лівосторонньому поданні (див. рисунок 8.4), інакше в правосторонньому (см. рисунок 8.5).

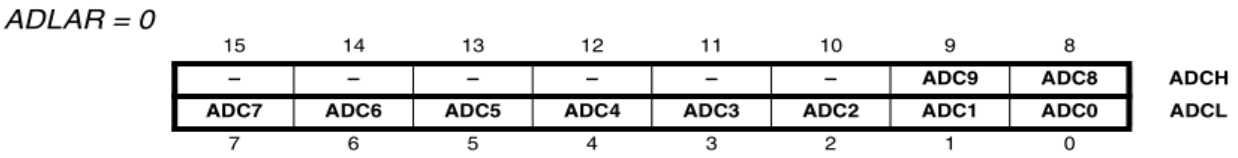


Рисунок 8.4 - Вирівнювання результату АЦП праворуч у регістрі ADC.

ADLAR = 1

15	14	13	12	11	10	9	8	
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	–	–	–	–	–	–	ADCL
7	6	5	4	3	2	1	0	

Рисунок 8.5 - Вирівнювання результату АЦП ліворуч у регістрі ADC.

Вирівнювання ліворуч рекомендується застосовувати якщо у проекті достатньо восьмибітного оцифрування сигналу, при цьому можна враховувати лише старші вісім біт ADC9:2, тобто регістр ADCH.

Bits 3:0 – MUX3:0: Analog Channel Selection Bits

Чотири біти MUX3:0 для ATmega328 (див. Table 23-4) і шість біт MUX5:0 для ATmega2560 (див. Table 129) визначають пін з якого буде зроблено зчитування аналогового сигналу, а також режим роботи (для ATmega2560). У таблиці 8.4 наведено збігаючі варіанти налаштування для пінів які існують як Arduino UNO, так і Arduino MEGA.

Таблиця 8.4. – Вибір каналу

MUX3...0	Номер піна
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

Режим одиночного перетворення

Як приклад розглянемо таке завдання:

за допомогою змінного резистора (потенціометра) змінювати яскравість свічення світлодіоду з використанням технології ШІМ.

Рішення.

Вибираємо мікроконтролер ATmega328.

Підключимо змінний резистор номіналом 10 кОм середнім виводом до піна ADC1=PC1=A1 (див. рисунок 8.6)

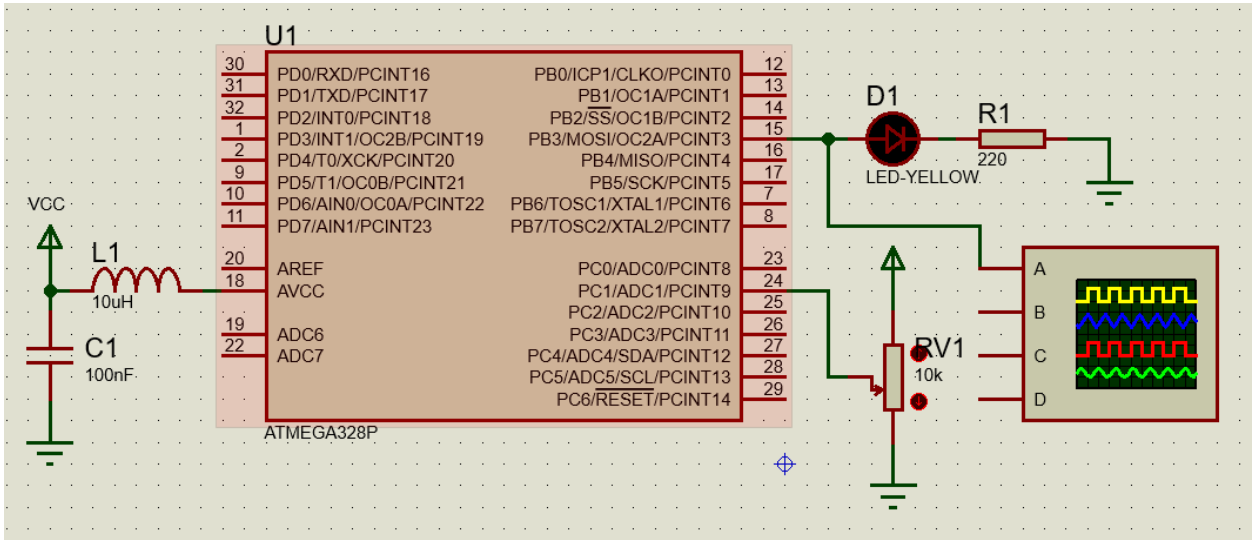


Рисунок 8.6 - Схема підключення потенціометру для використання АЦП

Для зміни яскравості світлодіоду задіємо другий таймер і ШІМ вивід OC2A=PB3=D11. Також для більш наочної візуалізації роботи підключимо осцилограф (Proteus) до цього піну.

Так як для ШІМ сигналу використовується другий таймер, то ШІМ сигнал може бути не більш ніж 8-бітний, отже, немає необхідності зчитувати 10 розрядів з модуля АЦП. Тому напишемо наступну функцію ініціалізації

```
void ADC_Init()
{
  ADCSRA |= (1<<ADEN)                 //Дозволяєм використовувати АЦП
  |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);    //Дільник 128 -> 125 кГц
  ADMUX |= (1<<REFS0)                      //ДОН 5V
  |(1<<MUX0)                               //вход ADC1 (PC1)
  |(1<<ADLAR);                        //ліве вирівнювання, тільки ADCH
}
```

Тут насамперед встановлюємо біт ADEN регістру ADCSRA для дозволу використання АЦП (див. пункт 23.9.2 технічної документації на ATmega328). Також у цьому регістрі встановимо дільник 128 бітами ADPS0:2 (Table 23-5). Бітами REFS0:1 регістру ADMUX вибираємо джерело опорної напруги (ДОН) яке дорівнює напруги живлення мікроконтролера (пункт 23.9.1 Table 23-3). Бітами MUX0:3 регістру ADMUX (Table 23-4) вибираємо пін ADC1 з якого зніматимемо аналоговий сигнал.

Також встановимо біт ADLAR регістру ADMUX для лівого вирівнювання оцифрованих даних (див. пункт 23.9.3.2). Це дозволить зчитувати лише старший байт ADCH 16-бітного регістру ADC.

Будемо використовувати одиночне АЦП перетворення, тому біт ADATE регістру ADCSRA залишимо рівним нулю за умовчанням.

Напишемо функцію читання аналогового сигналу.

```
uint8_t ADC_read()
{
  ADCSRA |= (1<<ADSC);          //START перетворення
  while((ADCSRA & (1<<ADSC))); //чекаємо поки перетворення не закінчилося
  return ADCH;
}
```

У нашому простому варіанті, функція не прийматиме жодних аргументів, а повертатиме оцифроване однобайтне не негативне значення, тому вказуємо тип `uint8_t`.

Перетворення починається установкою ADSC біта регістру ADCSRA (див. пункт 23.9.2). У цьому ж пункті написано, що цей біт дорівнюватиме одиниці поки перетворення не завершилося. Тому це можна використовувати для очікування кінця перетворення і як тільки цей біт буде модулем АЦП скинутий у нуль, то умова в циклі while стане хибним і станеться вихід із порожнього циклу. На наступному кроці зчитуємо ADCH – старший байт регістру ADC.

Щоб використовувати ШІМ сигнал на другому таймері, потрібно його налаштувати. Вибираємо режим фазової корекції №1 зі значенням TOP=255 (див. пункт 17.11.1 та Table 17-8).

```
TCCR2A |= (1<<WGM20); //режим PWM Phase Correct №1 TOP=255
```

Оскільки за схемою світлодіод підключений до піна PB3=OC2A, то налаштуємо канал А таймера №2 на неівертований ШІМ (див. Table 17-4).

```
  TCCR2A |= (1<<COM2A1); //неінвертований ШІМ
```

Запустимо таймер встановивши дільник рівним 32 за допомогою регістра TCCR2B (див. пункт 17.11.2) та бітів CS20:2 (див. Table 17-9).

```
  TCCR2B |= (1<<CS21)|(1<<CS20); //дільник 32
```

Отримаємо частоту ШІМ сигналу використовуючи формулу (3.14)

$$f_{OC0x} = \frac{f_{clk}}{2 \cdot N \cdot TOP} = \frac{16 \cdot 10^6}{2 \cdot 32 \cdot 255} = 980.39\,\text{Гц}. \quad (8.3)$$

Тоді період сигналу дорівнює

$$T = \frac{1}{f_{OC0x}} = \frac{1}{980.39} = 1.02\,\text{мс}. \quad (8.4)$$

Залишилося не забути налаштувати пін PB3 як ВИХІД

```
DDRB |= (1<<DDB3); //PB3=OC2A=D11 як ВИХІД
```

Наведемо повний код розробленої програми

Prog8.1

```
#define F_CPU 16000000UL
#include <avr/io.h>

void ADC_Init(){
  ADCSRA |= (1<<ADEN)                 // Дозволяєм використовувати АЦП
  |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);// Дільник 128 -> 125 кГц
  ADMUX |= (1<<REFS0)                 // ДОН 5V
  |(1<<MUX0)                          //вход ADC1 (PC1)
  |(1<<ADLAR);                        // ліве вирівнювання, тільки ADCH
}
uint8_t ADC_read(){
  ADCSRA |= (1<<ADSC);               //Старт перетворення
  while((ADCSRA & (1<<ADSC)));       //чекаємо закінчення перетворення
  return ADCH;
}
int main(){
   DDRB |= (1<<DDB3);               //PB3=OC2A=D11 як ВИХІД
   TCCR2A |= (1<<WGM20);            //режим PWM Phase Correct №1 TOP=255
   TCCR2A |= (1<<COM2A1);           //неінвертований ШІМ
   TCCR2B |= (1<<CS21)|(1<<CS20); //дільник 32
   ADC_Init();
   while (1){
      OCR2A = ADC_read();
   }
  return 0;
}
```

Після запуску **Prog8.1** отримуємо результат, представлений на рисунку 8.7.

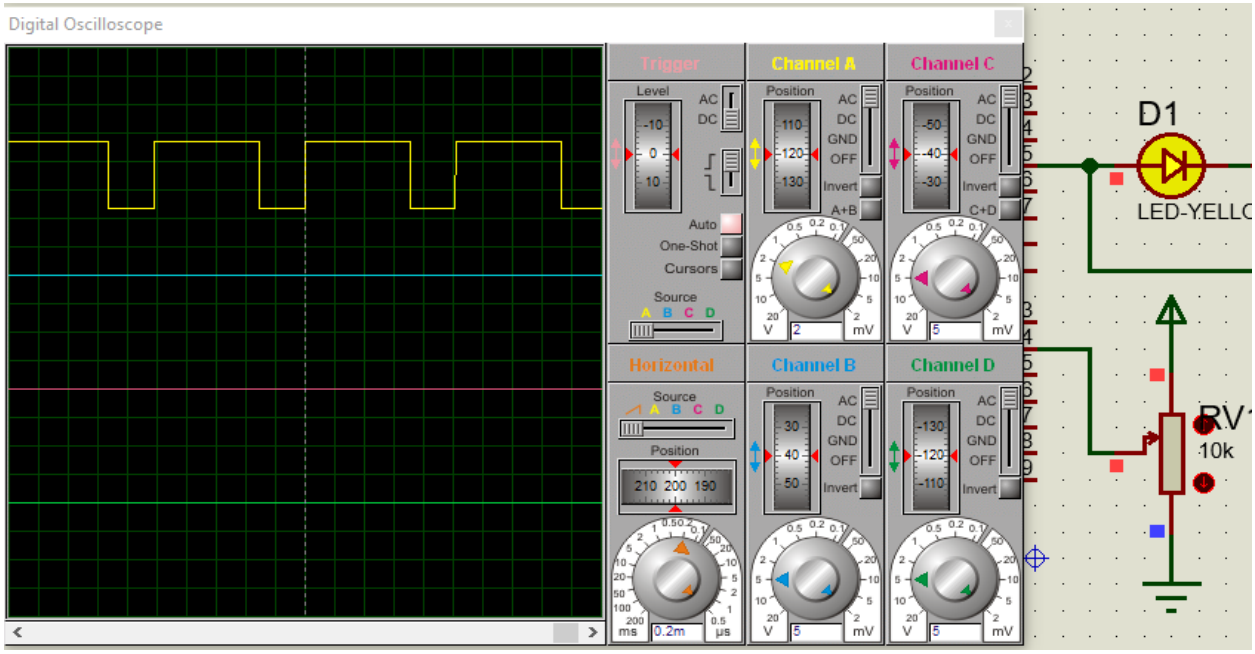


Рисунок 8.7 - Сигнал на піні PB3=OC2A=D11

З цього рисунка бачимо, що генерується ШІМ сигнал із періодом трохи більше 1 мс. Коефіцієнт заповнення сигналу пропорційний напрузі, яка

знімається зі змінного резистора. У крайньому верхньому положенні повзунка отримаємо завжди високий сигнал, а крайньому нижньому – низький.

Зазначимо один важливий момент. Розробник дозволяє визначити функцію читання іншим способом.

```
uint8_t ADC_read(){
  ADCSRA |= (1<<ADSC);              // Старт перетворення
  while(!(ADCSRA & (1<<ADIF)));     // чекаємо поки відбувається АЦП
  ADCSRA |= (1<<ADIF);              // скидаємо ADIF установкою 1
  return ADCH;
}
```

У цьому випадку для перевірки завершення перетворення модуля АЦП використовується біт ADIF регістру ADCSRA (див. пункт 23.9.2). Тут зазначається, що це біт встановлюється, коли перетворення АЦП завершується і регістри даних оновлюються. Якщо після цього викликається переривання, біт ADIF очищається апаратно. Очищення біта відбувається шляхом записування логічної одиниці у прапор.

Режим безперервного перетворення

Вирішимо тепер попереднє завдання, але з використанням АЦП в режимі безперервного перетворення. Для цього ми повинні увімкнути біт ADATE регістру ADCSRA (див. пункт 23.9.2). Установка цього біта включає режим (ADC Auto Trigger Enable) автоматичного запуску, причому АЦП почне перетворення по позитивному фронту обраного тригерного сигналу. Джерело запуску вибирається встановленням бітів вибору ADTS0:2 у регістрі ADCSRB. (Див. Table 23-6). За замовчуванням, коли всі ці біти дорівнюють нулю, вибирається режим (Free Running mode) безперервного перетворення. Біт ACME регістру ADCSRB включає режим аналогового компаратора, який нас зараз не цікавить, тому для надійності можна просто цьому регістру привласнити значення, що дорівнює нулю. Додаємо в функцію ініціалізації запуск АЦП установкою біта ADSC регістру ADCSRA (див. пункт 23.9.2) та отримаємо наступний код

```
void ADC_InitFM(){      // ініціалізація ADC free mode
  ADCSRA |= (1<<ADEN)   // Дозволяєм використовувати АЦП
  |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)   // Дільник 128 -> 125 кГц
```

```
  |(1<<ADATE);          // включаємо режим автоматичного запуску
  ADMUX |= (1<<REFS0)   // ДОН 5V
  |(1<<MUX0)            // вход ADC1 (PC1)
  |(1<<ADLAR);          // ліве вирівнювання, тільки ADCH
  ADCSRB = 0;           // режим безперервного АЦП
  ADCSRA |= (1<<ADSC); // Починаємо перше перетворення
}
```

Так як зараз АЦП працює в безперервному режимі, то немає необхідності створювати функцію запуску, можна перевірити чи завершене перетворення і зчитати регістр ADC

```
if(ADCSRA & (1<<ADIF)){    // якщо перетворення завершено
    ADCSRA |= (1<<ADIF);  //скидаємо ADIF установкою 1
    OCR2A =  ADCH;    //зчитуємо значення АЦП у регістр порівняння
 }
```

Остаточно отримуємо наступну програму

Prog8.2

```
1    #define F_CPU 16000000UL
2    #include <avr/io.h>
3
4    void ADC_InitFM(){
5      ADCSRA |= (1<<ADEN)  // Дозволяєм використовувати АЦП
6      |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)  // Дільник 128 -> 125 кГц
7      |(1<<ADATE);          // включаємо режим автоматического запуска
8      ADMUX |= (1<<REFS0)  // ДОН 5V
9      |(1<<MUX0)            //вход ADC1 (PC1)
10     |(1<<ADLAR);          // ліве вирівнювання, тільки ADCH
11     ADCSRB = 0;           // режим безперервного АЦП
12     ADCSRA |= (1<<ADSC); // Починаємо перетворення
13   }
14   int main(){
15      DDRB |= (1<<DDB3);              //PB3=OC2A=D11 як ВИХІД
16      TCCR2A |= (1<<WGM20);           //режим PWM Phase Correct №1 TOP=255
17      TCCR2A |= (1<<COM2A1);          //неінвертований ШІМ
18      TCCR2B |= (1<<CS21)|(1<<CS20); //дільник 32
19      ADC_InitFM();
20      while (1){
21         if(ADCSRA & (1<<ADIF)){       // якщо перетворення завершено
22           ADCSRA |= (1<<ADIF);        // скидаємо ADIF установкою 1
23           OCR2A =  ADCH;              //зчитуємо значення АЦП у регістр порівняння
24         }
25      }
26    return 0;
27   }
```

Використання вбудованого переривання у модулі АЦП

Як неодноразово зазначалося використання переривань дозволяє менше навантажувати основну програму шляхом винесення деяких частин коду в обробники переривань. Більш того використання переривань дозволяє не витрачати час і ресурси центрального процесора на очікування завершення того чи іншого повільного процесу. Зокрема, в даній лабораторній роботі ми, використовуючи переривання від модуля АЦП, не витрачатимемо час на очікування завершення процесу оцифровування аналогового сигналу.

Отже, знаходимо переривання АЦП у Table 11-1 для мікроконтролера ATmega328 та Table 32 для ATmega2560. Таке переривання має одну назву ADC і викликається при завершенні перетворення (ADC Conversion Complete).

Для використання переривання необхідно:

1) підключити файл бібліотеки переривань interrupt.h;

2) дозволити локально переривання ADC;

3) дозволити глобальні переривання;

4) описати обробник переривання.

Перший пункт виконується командою

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

Для виконання другого пункту необхідно встановити біт ADIE регістру ADCSRA (див. пункт 23.9.2)

```
 ADCSRA |= (1<<ADIE); // Дозвіл переривання АЦП
```

Глобальні переривання стандартно дозволяються командою

```
sei();
```

Відповідно до матеріалу п’ятої лабораторної роботи, обробник переривання в даному випадку описуватиметься так

```
ISR(ADC_vect){        //обробник переривання
   //тіло обробника переривання
}
```

Об'єднуючи всю цю інформацію маємо програму **Prog8.3.**

Prog8.3

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

void ADC_InitFM_interrupt(){
  ADCSRA |= (1<<ADEN)  // Дозволяєм використовувати АЦП
  |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)  // Дільник 128 -> 125 кГц
  |(1<<ADATE)          // включаємо режим автоматичного запуску
  |(1<<ADIE);          // Дозволяєм переривання
  ADMUX |= (1<<REFS0)  // ДОН 5V
  |(1<<MUX0)           //вход ADC1 (PC1)
  |(1<<ADLAR);         // ліве вирівнювання, тільки ADCH
  ADCSRB = 0;          // режим безперервного АЦП
  ADCSRA |= (1<<ADSC); // Починаємо перше перетворення
}
int main(){
   DDRB |= (1<<DDB3);             //PB3=OC2A=D11 як ВИХІД
   TCCR2A |= (1<<WGM20);          //режим PWM Phase Correct №1 TOP=255
   TCCR2A |= (1<<COM2A1);         //неінвертований ШІМ
   TCCR2B |= (1<<CS21)|(1<<CS20); //дільник 32
   ADC_InitFM_interrupt();
   sei();
   while (1){};
 return 0;
}
ISR(ADC_vect){        //обробник переривання
    OCR2A =  ADCH;    //зчитуємо значення АЦП у регістр порівняння
}
```

Функція ініціалізації налаштовує модуль АЦП на безперервне перетворення, дозволяє переривання і включає запуск перетворення. Основна програма порожня, вся обробка відбувається у перериванні. Як тільки перетворення АЦП завершується управління передається у вектор переривання, в якому значення регістру АЦП передається в регістр порівняння каналу А другого таймера.

Візуалізація результату АЦП за допомогою UART

Розглянемо наступний приклад. Будемо виводити отримані оцифровані значення в Virtual Terminal, якщо ми працюємо в Proteus і Serial Monitor - якщо в Arduino IDE.

На початку вирішимо це завдання на платформі Arduino MEGA. Зберемо схему, зображену рисунку 8.8.

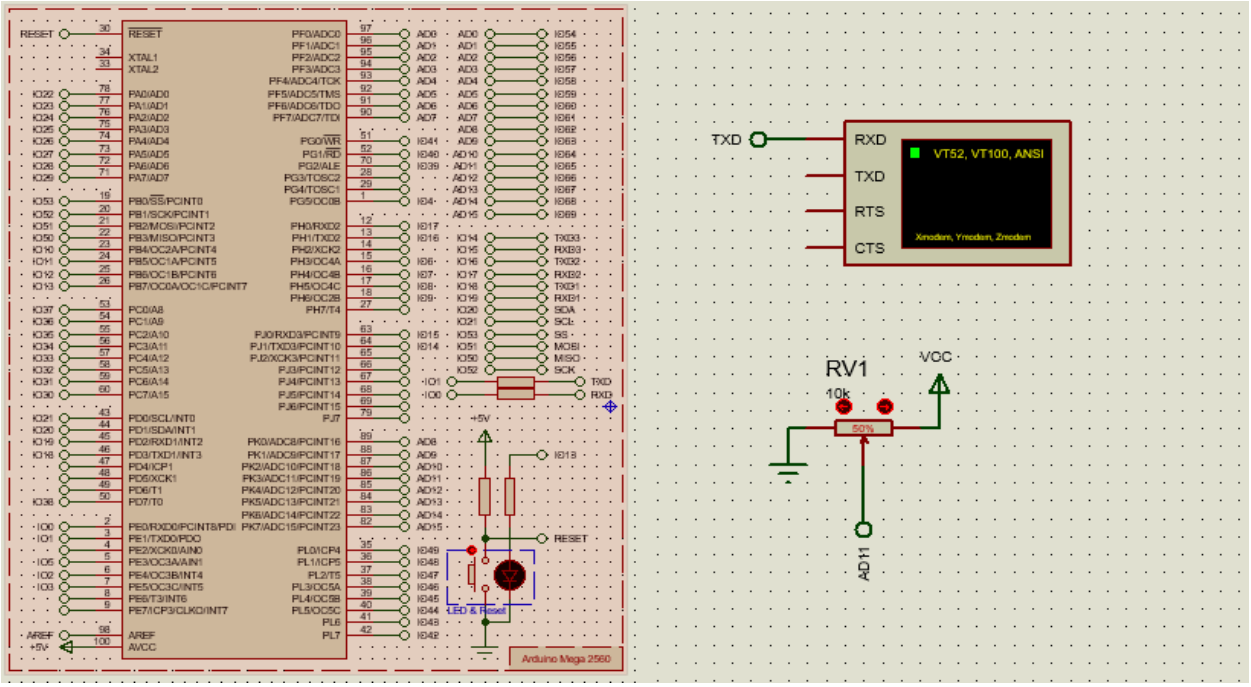


Рисунок 8.8 - Підключення потенціометра та Virtual Terminal до Arduino MEGA

Відповідно до цієї схеми потенціометр на 10 кОм (елемент pot-hg) двома крайніми виводами підключається до «+» і «-» живлення, а середній вивід, з якого знімається напруга, підключений до аналогового входу A11 в ідеології платформи Arduino або до виводу ADC11 згідно з описом розробника. Virtual Terminal будемо використовувати тільки на прийом тому задіємо його RXD вивід, який підключимо до UART0 (для узгодження з реальною платформою) на пін TXD = D1.

Відмінностей у модулях АЦП мікроконтролерів ATmega328 та ATmega2560 досить багато, проте той функціонал, яким ми користуватимемося в рамках лабораторних робіт, практично ідентичний у цих двох МК. Так регістри ADCSRA повністю збігаються (див. сторінку 297 документації на ATmega2560). Бітове поле ADPS2:0 цього регістру відповідає за налаштування дільника (див. Table 130) і повністю еквівалентне ATmega328. Регістр ADCSRB ATmega2560 має додатковий біт MUX5, який розташований на третьому місці (див. сторінку 295). Бітове поле ADTS2:0 цього регістру відповідає за вибір автоматичного режиму (див. Table 131) та повністю еквівалентно з ATmega328. Регістр ADMUX ATmega2560 містить додатковий біт MUX4, розташований на четвертому місці (див. сторінку 294). Бітове поле REFS1:0 цього регістру відповідає за вибір

опорної напруги (див. Table 128) та має одну додаткову можливість порівняно з ATmega328. Однак використовуваний нами режим опорної напруги, яка дорівнює напрузі живлення мікроконтролера налаштовується аналогічно ATmega328. Принципово відрізняються режими роботи в ATmega328 та ATmega2560, які налаштовуються бітами MUX5:0. Порівняйте Table 129 для ATmega2560 та Table 23-4 для ATmega328. З Table 129 знаходимо, що для використання піна ADC11 як входу аналогового сигналу біти MUX5 = MUX1 = MUX0 = 1 повинні бути встановлені в одиницю, а біти MUX4 = MUX3 = MUX2 = 0 - нуль. Враховуючи все вище сказане, отримуємо наступну програму

Prog8.4

```
1                                //ATmega2560
2    #define F_CPU 16000000UL
3    #include <avr/io.h>
4    #include <avr/interrupt.h>
5
6    volatile uint16_t adc;    // змінна для зберігання результату АЦП
7
8    void ADC_InitFM_interrupt(){
9      ADCSRA |= (1<<ADEN)          // Дозволяєм використовувати АЦП
10     |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)  // Дільник 128 -> 125 кГц
11     |(1<<ADATE)                  //включаємо режим автоматичного запуску
12     |(1<<ADIE);                  // Дозволяєм переривання
13     ADMUX |= (1<<REFS0)          // ДОН 5V
14     |(1<<MUX1)|(1<<MUX0);        //вход ADC11=A11=AD11=PK3 +MUX5 в ADCSRB
15       //праве вирівнювання ADLAR=0, режим за замовчуванням, зчитуємо ADC
16     ADCSRB |= (1<<MUX5); //+ режим безперервного перетворення за замовчуванням
17     ADCSRA |= (1<<ADSC);         //Старт перетворення
18   }
19   int main(){
20      ADC_InitFM_interrupt();
21      sei();
22      Serial.begin(9600);
23      while (1){
24        Serial.print("ADC = ");
25        Serial.println(adc);
26      }
27    return 0;
28   }
29   ISR(ADC_vect){      //обробник переривання
30       adc =  ADC;     // зчитуємо значення АЦП у змінну adc
31   }
```

Натиснувши на симуляцію (або запустивши програму на реальному устаткуванні), отримаємо у Virtual Terminal дані від 0 до 1023, залежно від положення ручки потенціометра.

У цій програмі для виведення значень протоколу UART були використані «ардуїновські» функції Serial.begin() і Serial.print() (див. рядки 22, 24, 25). Але

матеріал сьомої лабораторної роботи дозволяє написати подібну програму по «дорослому» на чистому СІ. Зберемо схему в Proteus на «голому» ATmega2560 зображену на рисунку 8.9. Підключення піна AVCC відповідає рисунку 122 технічної документації.

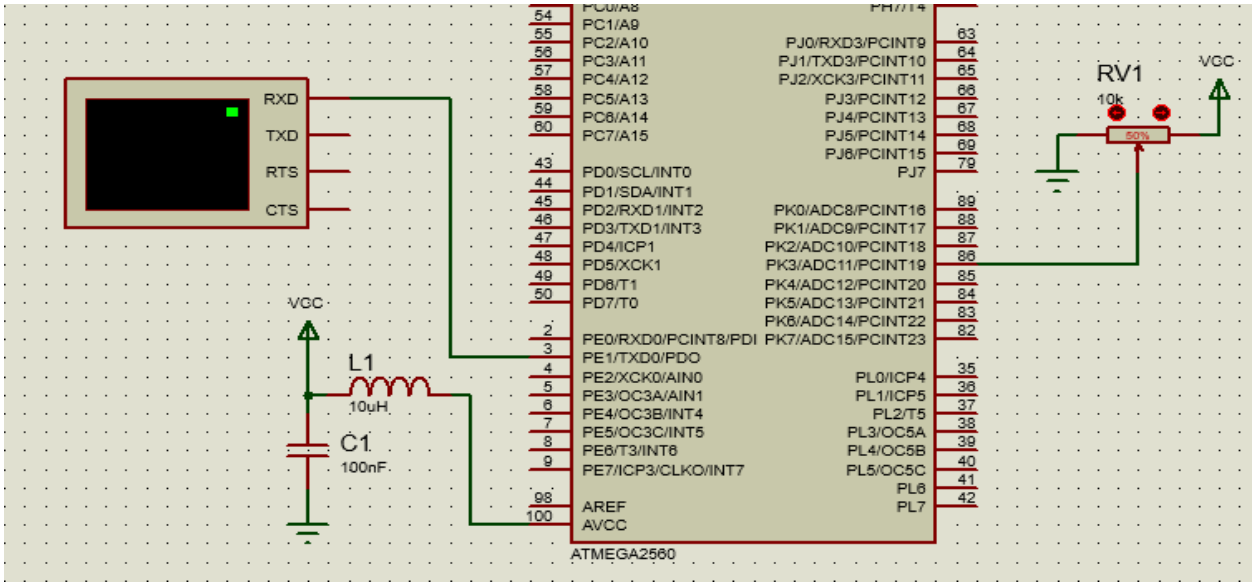


Рисунок 8.9 - Підключення потенціометра та Virtual Terminal до ATmega2560

Виберемо середу розробки в Proteus WinAVR та наберемо програму **Prog8.5**

Prog8.5

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

volatile uint16_t adc;   // змінна для зберігання результату АЦП

void ADC_InitFM_interrupt(){
  ADCSRA |= (1<<ADEN) // Дозволяєм використовувати АЦП
  |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)  // Дільник 128 -> 125 кГц
  |(1<<ADATE)         // включаємо режим автоматичного запуску
  |(1<<ADIE);         // Дозволяєм переривання
  ADMUX |= (1<<REFS0)          // ДОН 5V
  |(1<<MUX1)|(1<<MUX0);        //вход ADC11=A11=AD11=PK3 +MUX5 в ADCSRB
                        // праве вирівнювання, ADLAR=0 зчитуємо ADC
  ADCSRB |= (1<<MUX5); //+ режим безперервного перетворення за замовчуванням
  ADCSRA |= (1<<ADSC);         //Старт перетворення
}
// функція ініціалізації UART
void UART_init(){
  // налаштування швидкості передачі
  UBRR0H = 0;
  UBRR0L = 103; // (16000000/(9600*16))-1=103 Table 1.6.5

  // налаштування
  //UCSR0A  U2Xn=0 одинарна швидкість передачі

  //UCSR0B
  //RXCIE0=TXCIE0=UDRIE0=0   не використовуємо переривання
```

```
30     //RXEN0=0                   не використовуємо прийом даних
31     UCSR0B |= (1 << TXEN0);     // дозволити передачу
32     //UCSZ02=0                  задає кількість біт, що передаються
33
34     //UCSR0C
35     //UMSEL01=UMSEL00=0         асинхронний режим
36     //UPM01=UPM00=0             контроль парності відсутній
37     //USBS0=0                   стоповий біт один
38     //UCSZ01=UCSZ00=1           з урахуванням UCSZ02 передаємо 8 біт
39   }
40   // Функція передачі одного символу
41   void UART_transmit_char( uint8_t c ){
42      // чекаємо поки буфер передачі не звільниться
43       while( !(UCSR0A & (1 << UDRE0)) );
44      UDR0 = c;
45   }
46
47   // Функція передачі рядка
48   void UART_transmit_str(char *str){
49       uint8_t c;
50        while((c =*str++)!= 0){// передаємо символи послідовно
51           UART_transmit_char(c);
52         }
53   }
54   // Функція передачі однієї цифри d
55   void UART_transmit_digit( uint8_t d ){
56      // чекаємо поки буфер передачі не звільниться
57      while( !(UCSR0A & (1 << UDRE0)) );
58      UDR0 = d+0x30;            // формуємо код ASCII заданої цифри
59   }
60   // Функція передачі цілого беззнакового числа D
61   void UART_transmit_uint( uint16_t D ){
62      uint8_t M[6], i=0;//В масиві зберігається D-у кожній комірці цифра
63       do {
64         M[i]=D%10;             // записуємо наймолодший розряд
65         D/=10;                 // відкидаємо від числа молодший розряд
66         i++;                   // збільшуємо лічильник цифр на 1
67       }
68       while (D>0);             // якщо D=0 усі цифри числа перебрали
69       for (;i>0;i--){          // починаємо цикл з i рівною кількістю цифр
70         UART_transmit_digit(M[i-1]);//передаємо цифри зі старшого розряду
71       }
72   }
73   int main(){
74      ADC_InitFM_interrupt();       // ініціалізуємо АЦП
75      sei();
76      UART_init();                  // ініціалізуємо UART
77      while (1){
78        UART_transmit_str("ADC = ");
79        UART_transmit_uint(adc);    //передаємо результат з АЦП в UART
80        UART_transmit_str("\r\n");
81        _delay_ms(300);
82      }
83    return 0;
84   }
85   ISR(ADC_vect){     //обробник переривання
86       adc =  ADC;    // зчитуємо значення АЦП у змінну adc
87   }
```

Функціонал програм **Prog8.4** та **Prog8.5** повністю еквівалентний. Усі функції роботи з протоколом UART взято з лабораторної роботи №7 і там

повністю розглянуто. Програма **Prog8.5** також буде працездатною середовищі Arduino IDE.

Хід роботи

Завдання 8.1.

Підключити три світлодіоди через необхідні резистори до зазначених пінів (P1, P2, P3), а також змінний резистор на аналоговий вхід Ax. Використовуючи режим одиночного перетворення для модуля АЦП, написати програму, яка в залежності від положення ручки потенціометра буде включати різну кількість світлодіодів: від нуля до трьох. Для ATmega328 зчитувати 10-бітне значення АЦП, для ATmega2560 – восьми бітове значення.

Переривання не використовувати.

Таблиця 8.5. – Вхідні данні для завдання 8.1

Варіант	Ax	P1	P2*	P3
1	A5	D13	A0	D2
2	A4	D12	A1	D3
3	A3	D11	A2	D4
4	A2	D10	A3	D5
5	A1	D9	A4	D6
6	A0	D8	A5	D7
7	A5	D2	A0	D8
8	A4	D3	A1	D9
9	A3	D4	A2	D10
10	A2	D5	A3	D11
11	A1	D6	A4	D12
12	A0	D7	A5	D13
13	A5	D13	A1	D4
14	A4	D12	A2	D5
15	A3	D11	A0	D6
16	A2	D10	A4	D7
17	A1	D9	A5	D3
18	A0	D8	A3	D4
19	A4	D7	A0	D12
20	A3	D6	A1	D10

* - пін P2 використовувати як цифровий.

Завдання 8.2.

Використовуючи режим безперервного аналого-цифрового перетворення написати програму керуючу двома світлодіодами, підключеними через необхідні резистори до ШІМ пін одного таймера.

Для ATmega328 зчитувати 10-бітне значення АЦП і використовувати 16-бітний таймер в режимі 10-бітного ШІМ з фазовою корекцією.

Для ATmega2560 зчитувати 8-бітове значення АЦП і використовувати 8-бітовий таймер в режимі 8-бітного ШІМ з фазовою корекцією.

Змінний резистор підключити до вказаного аналогового піна. У крайньому лівому положенні ручки потенціометра обидва світлодіоди повинні бути вимкнені. При повороті ручки потенціометра вправо перший світлодіод повинен поступово збільшувати свою яскравість і при повороті на 50% повинен світитися на повну силу. При подальшому повороті ручки потенціометра другий світлодіод повинен поступово розгорятися, при цьому перший світиться на 100%. У крайньому правому положенні ручки змінного резистора обидва світлодіоди повинні світитися на максимальній яскравості. Обертання ручки вліво повинно призвести до поступового зменшення яскравості спочатку другого світлодіоду, а потім першого і в крайньому лівому положенні обидва світлодіоди повинні згаснути.

Переривання не використовувати.

Таблиця 8.6. – Вхідні данні для завдання 8.2

Варіант	Ax	Варіант	Ax
1	A2	11	A3
2	A3	12	A2
3	A5	13	A1
4	A4	14	A0
5	A3	15	A5
6	A2	16	A4
7	A1	17	A3
8	A0	18	A2
9	A5	19	A1
10	A4	20	A0

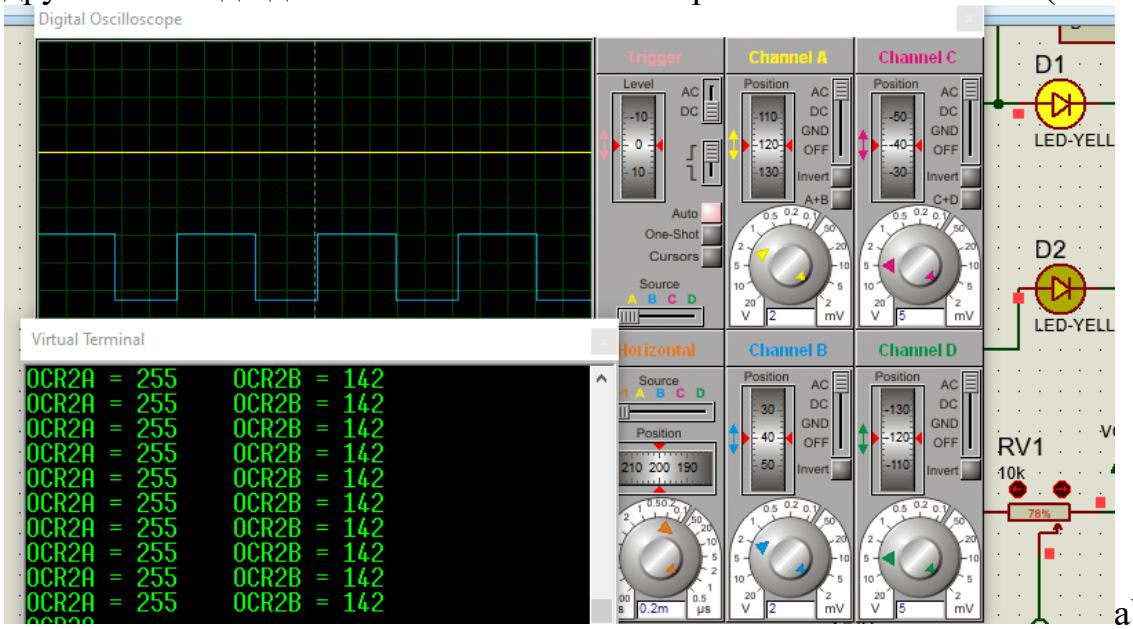
Примітка: Proteus не завжди правильно відображає яскравість світлодіоду в залежності від сигналу ШИМ. Тому підключіть паралельно світлодіодам два канали віртуального осцилографа та переконайтеся, що генеруються необхідні сигнали. Надайте у звіті відповідні скріншоти.

Завдання 8.3.

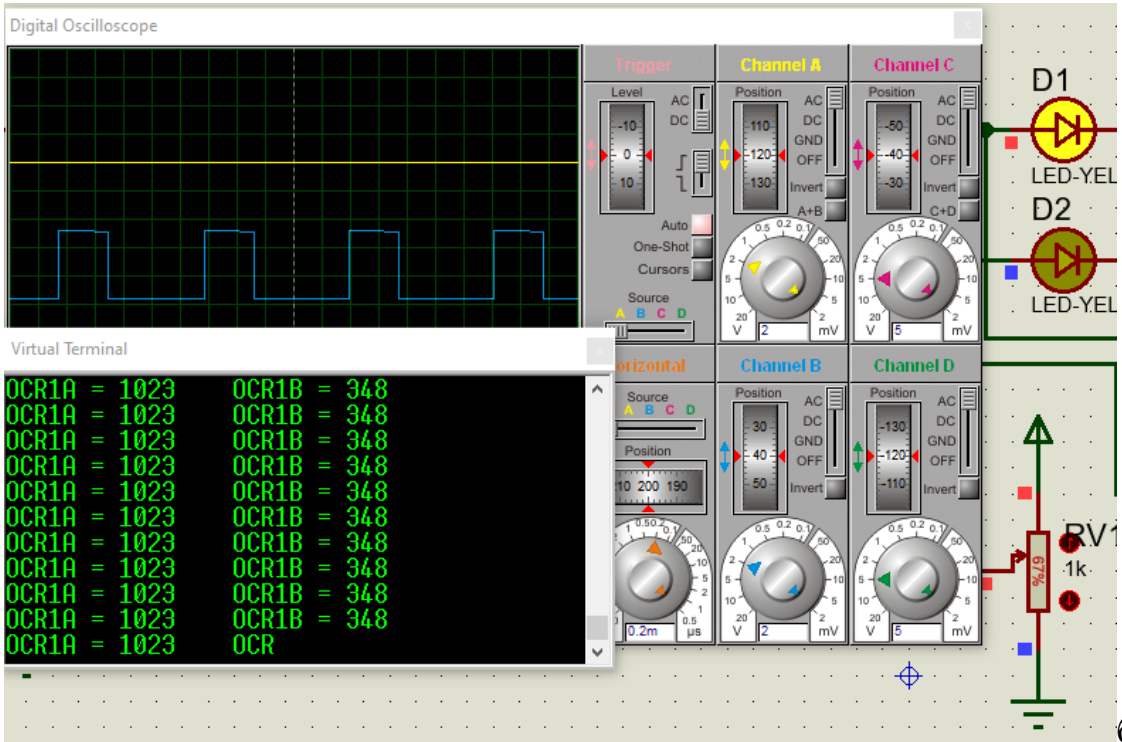
Повторити завдання 8.2, але для іншого мікроконтролера з одночасним виведенням у Virtual Terminal або Serial Monitor інформації про поточні значення використовуваних регістрів порівняння.

Використовувати вбудоване переривання модуля АЦП.

Для прикладу дивись малюнок, на якому видно, що повзунок резистора займає положення 78% (67%), при цьому перший світлодіод світиться з максимальною яскравістю, що відповідає OCR2A = 255 (OCR1A = 2023), а другий світлодіод світиться з неповною яскравістю OCR2B = 142 (OCR1B = 348).



а)



б)

Рисунок 8.10 - Осцилограми в залежності від ШІМ параметра

Verweise / References / Література

1. Офіційний технічний опис виробника на мікроконтролер ATmega328P https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
2. Офіційний технічний опис виробника на мікроконтролер ATmega2560 https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
3. Програма Proteus https://labcenter.s3.amazonaws.com/downloads/Tutorials.pdf
4. Brian W. Kernighan, Dennis M. Ritchie The C Programming Language: AT&T Bell Laboratories Murray Hill, New Jersey, 1988, 272 p.
5. Massimo Banzi, Michael Shiloh, Getting Started with Arduino, 3rd Edition, United States of America: Maker Media, 2014.

SCIENTIFIC EDITION

MONOGRAPH

INTELLEKTUELLES KAPITAL - DIE GRUNDLAGE FÜR INNOVATIVE ENTWICKLUNG

ROBOTISCHE SYSTEME

INTELLECTUAL CAPITAL IS THE FOUNDATION OF INNOVATIVE DEVELOPMENT

ROBOTIC SYSTEMS

MONOGRAPHIC SERIES «EUROPEAN SCIENCE»

BOOK 28. PART 1

Authors / Автори:
Albert Kotvytskiy / *Котвицький А. Т.*

The scientific achievements of the authors of the monograph were also reviewed and recommended for publication at the international scientific symposium
«Intellektuelles Kapital - die Grundlage für innovative Entwicklung /
Intellectual capital is the foundation of innovative development '2024»
(March 30, 2024)

Monograph published in the author's edition

The monograph is included in
International scientometric databases

500 copies
March, 2024

Published:
ScientificWorld -NetAkhatAV
Lußstr 13,
Karlsruhe, Germany

e-mail: editor@promonograph.org
https://desymp.promonograph.org

ISBN 978-3-989240-41-4