МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА АВІАЦІЙНИХ КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ КОМПЛЕКСІВ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускної кафедри
_____В. М. Синєглазов
«___»_____2021 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗА СПЕЦІАЛЬНІСТЮ 151 «АВТОМАТИЗАЦІЯ І КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ»

Тема: «Універсальне автоматизоване робоче місце розробника програмного забезпечення мікроконтролерів»

Виконавець: студент 435 групи Савчук Сергій Юрійович

Керівник: професор Сергеєв Ігор Юрійович

Нормоконтролер: _____ _____

КИЇВ 2021

ЗМІСТ

ВСТУП

Актуальність розробки автоматизованого робочого місця полягає у тому, що обсяги інформації у наш час лише збільшуються, і виникає нагальна потреба у їх вчасному та компетентному опрацюванні. Важливо максимально спростити контакт з автоматичними системами керування та підготувати фундаментальну базу для роботи з ними. Освоєння написання простих програм зможе в подальшому забезпечити спеціалістам можливість створення кодів більш складного рівня.

Мета і завдання виконання дипломної роботи полягає у аналізі одних із найпоширеніших серій мікроконтролерів (а саме: КР 580, PIC, AVR, STM) та розробці універсального автоматизованого робочого місця з програмування найбільш вдалого з них.

Об'єкт дослідження: програмне середовище для розробки універсального автоматизованого робочого місця.

Предмет дослідження: мікроконтролери КР 580, PIC, AVR, STM.

У процесі дослідження було застосовано емпіричні (порівняння, вимірювання, експеримент) та комплексні (аналіз, моделювання) методи.

Наукова новизна отриманих результатів полягає у глибшому аналізі виокремлених серій мікроконтролерів з подальшим визначенням їхніх суттєвих переваг та недоліків при створенні оптимального робочого місця.

Практичне значення отриманих результатів полягає у розробці автоматизованого робочого місця та програмного забезпечення для нього, виділенні необхідних компонентів для моделювання лабораторного стенду з вивчення мікроконтролерів.

РОЗДІЛ І. АРХІТЕКТУРА СУЧАСНИХ МІКРОКОНТРОЛЕРІВ

1.1. Загальні відомості про мікроконтролери

Мікроконтролер (рисунок 1) - мікросхема, яка дозволяє керувати різними електронними пристроями, містить в собі ПЗУ (постійно запам’ятовуючий пристрій) і ОЗУ (оперативно запам'ятовуючий пристрій). По суті, мікроконтролер є однокристальним комп'ютером, здатним виконувати прості операції. Він має в наявності інтегровані порти введення-виведення, таймери та інші периферійні пристрої.

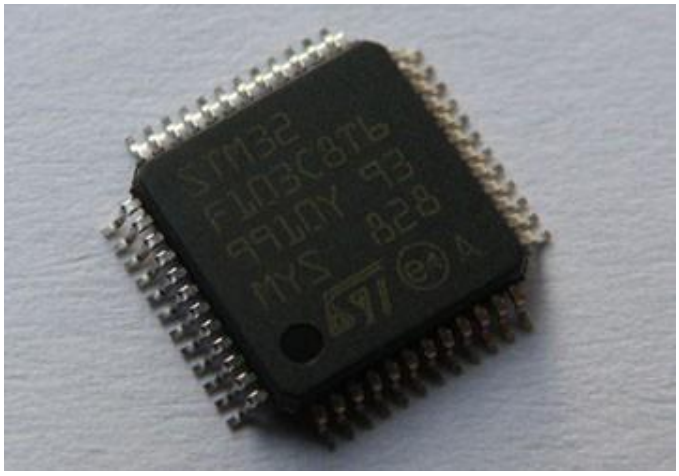


Рис. 1. Мікроконтролер фірми STMicroelectronics

Мікроконтролери застосовують майже скрізь. Практично кожен пристрій в XXI столітті працює на мікроконтролері: вимірювальні прилади, інструменти, побутова техніка, годинники, іграшки, музичні шкатулки та листівки, а також багато іншого.

Розробник може використовувати аналоговий сигнал подова його на вхід мікроконтролера і маніпулювати з даними про його значення. Цю роботу

виконує аналогово-цифровий перетворювач (АЦП). Ця функція дозволяє спілкуватися користувачеві з мікро контролером, а також сприймати різні параметри навколишнього світу за допомогою датчиків.

Програмне забезпечення для мікроконтролерів пишеться на спеціальних мовах програмування (зазвичай на мові Асемблера або Сі) і записуються в пам'ять контролера за допомогою вибору програм (рисунок 2). Програматор - це апаратно-програмний пристрій, призначений для зчитування і запису в постійний запам'ятовуючий пристрій.

Для зберігання виконуваної програми в мікроконтролері використовується пам'ять FLASH, так звана пам'ять програми. FLASH-пам'ять є енергозалежною, і може бути переписана близько десяти тисяч разів. Дана пам'ять є подобою жорсткого диску персонального комп'ютера.

Для зберігання різних змінних даних, які використовуються під час виконання програм, застосовують оперативну пам'ять, також відому як RAM. Ця пам'ять може бути записана і стерта будь-яку кількість разів, але дані зберігаються в ній до тих пір, поки мікроконтролер підключений до живлення. Є подобою оперативної пам'яті персонального комп’ютера.

Рис. 2. Програматор фірми STMicroelectronics

Для зберігання даних, які рідко змінюються (і збереження даних під час того, як мікроконтролер відключений) використовується пам'ять EEPROM. У деяких мікроконтролерах даний тип пам'яті відсутній, тому доводиться емулювати його програмно.

При проектуванні мікроконтролерів доводиться дотримуватися компромісу між розмірами і вартістю з одного боку, і гнучкістю та продуктивністю - з іншого. Для різних додатків оптимальне співвідношення цих та інших параметрів може відрізнятися дуже сильно. Тому існує величезна кількість типів мікроконтролерів, що відрізняються архітектурою процесорного модуля, розміром і типом вбудованої пам'яті, набором периферійних пристроїв, типом корпусу і т. д. На відміну від звичайних комп’ютерних мікропроцесорів, в мікроконтролерах часто використовується гарвардська архітектура пам'яті, тобто роздільне зберігання даних і команд в ОЗУ і ПЗУ відповідно.

Мікроконтролери часто містять вбудовані периферійні пристрої, які дозволяють використовувати одну маленьку мікросхему замість великої кількості окремих пристроїв.

Неповний список периферійних пристроїв, які можуть використовуватися в мікроконтролерах, включає в себе:

– універсальні цифрові порти, які можна налаштовувати як на введення, так і на виведення;

– різні інтерфейси введення-виведення, такі, як UART, I²C, SPI, CAN, USB, IEEE 1394, Ethernet;

– аналого-цифрові і цифро-аналогові перетворювачі;

– компаратори;

– широтно-імпульсні модулятори (ШІМ-контролер);

– таймери;

– контролери безколекторних двигунів, в тому числі крокових;

– контролери дисплеїв і клавіатур;

– радіочастотні приймачі і передавачі;

– масиви вбудованої FLASH-пам'яті;

– вбудовані тактовий генератор і сторожовий таймер.

Для продовження теми мікроконтролерів необхідно ввести ряд визначень:

1. Порти введення-виведення (GPIO) - групи висновків мікроконтролерів, призначених для спілкування мікроконтролера з «зовнішнім мікро».

2. UART, I2C, SPI, CAN, USB - різні інтерфейси введення-виведення, призначені для передачі даних.

3. АЦП – аналогово-цифровий перетворювач, призначений для вимірювання значень вхідних напруг і перетворення їх на цифрову інформацію.

4. ЦАП – цифро-аналоговий перетворювач, призначений для перетворення вхідного цифрового сигналу на вихідний аналоговий сигнал.

5. Таймер - дозволяє дуже точно відміряти інтервали часу, генерувати вихідний ШІМ-сигнал, підраховувати період вхідного сигналу і т. д.

6. Watchdog - сторожовий таймер, призначений для контролю над зависанням системи.

7. DMA - модуль прямого доступу до пам'яті, дозволяє передавати дані між пристроями, не використовуючи ресурси процесора.

Застосування мікроконтролерів в сучасному світі є дуже широким, так як використовується досить потужний обчислювальний пристрій з широким спектром можливостей, побудований всього на одній мікросхемі, що в значній мірі знижує енергоспоживання і вартість пристроїв, побудованих на його базі.

Мікроконтролери використовують для управління різними електронними пристроями і окремими блоками:

– в обчислювальній техніці - контролери для жорстких / гнучких дисків, в материнських платах, калькуляторах;

– в електроніці і різноманітних пристроях побутової техніки, в якій використовуються електронні системи управління - мікрохвильових печах,

пральних машинах, телефонах, різних роботах, в системах «Розумний будинок».

Мікроконтролери також знайшли широке застосування в промисловості:

– пристрої промислової автоматики - від програмованого реле до ПЛК;

– системи управління верстатами.

Програмне забезпечення мікроконтролера - невід'ємна частина системи, без якої апаратні засоби нежиттєздатні, більш того, кінцевий користувач системи, якому зазвичай важливий функціонал, може навіть не здогадуватися про наявність в ній програмно-керованих компонентів.

1.2. Огляд мікроконтролерів серій KP 580, PIC, AVR, STM

AVR - це назва популярного сімейства мікроконтролерів, яке виготовляє компанія Atmel. Окрім AVR під цим брендом випускають мікроконтролери й інших архітектур, наприклад, ARM та i8051.

Існує три види цих мікроконтролерів:

1. AVR 8-bit.
2. AVR 32-bit.
3. AVR xMega

Найпопулярнішим за останні роки є саме 8-бітове сімейство мікроконтролерів. З них розпочиналося виготовлення таких простих виробів, як світлодіодні індикатори, термометри, годинники, а також проста автоматика типу управління освітленням і нагрівальних приладів.

Мікроконтролери AVR 8-bit, у свою чергу, поділяються на два популярних сімейства: Attiny та Atmega. Attiny здебільшого мають від 8 пінів і більше. Обсяг їхньої пам'яті і функціонал зазвичай скромніше, ніж в наступних, більш удосконалених мікроконтролерів – Atmega. Вони мають більший обсяг пам'яті, висновків і різних функціональних вузлів.

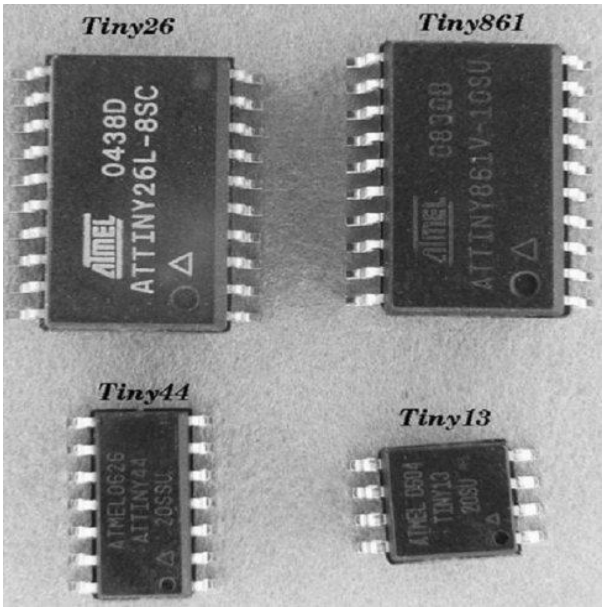


Рис. 3. Мікроконтролери серії AVR

Найпотужнішим підвидом мікроконтролерів є xMega. Ці мікроконтролери випускаються в корпусах з величезною кількістю пінів - від 44 до 100 - стільки необхідно для створення проектів з великою кількістю датчиків і виконавчих механізмів. Окрім того, збільшений об'єм пам'яті і швидкість роботи дозволяють отримати високу швидкодію.

У поширених AVR-мікроконтролерах, таких як Atmega328, використовують 8 канальний АЦП, з розрядністю 10 біт. Це означає, що користувач зможе зчитати значення з 8 аналогових датчиків. До цифрових виводів підключаються цифрові датчики. Однак цифровий сигнал може бути тільки 1 (одиницею) або 0 (нулем), в той час як аналоговий може відображати безліч значень.

Загалом структуру AVR-мікроконтролера можна зобразити наступним чином:

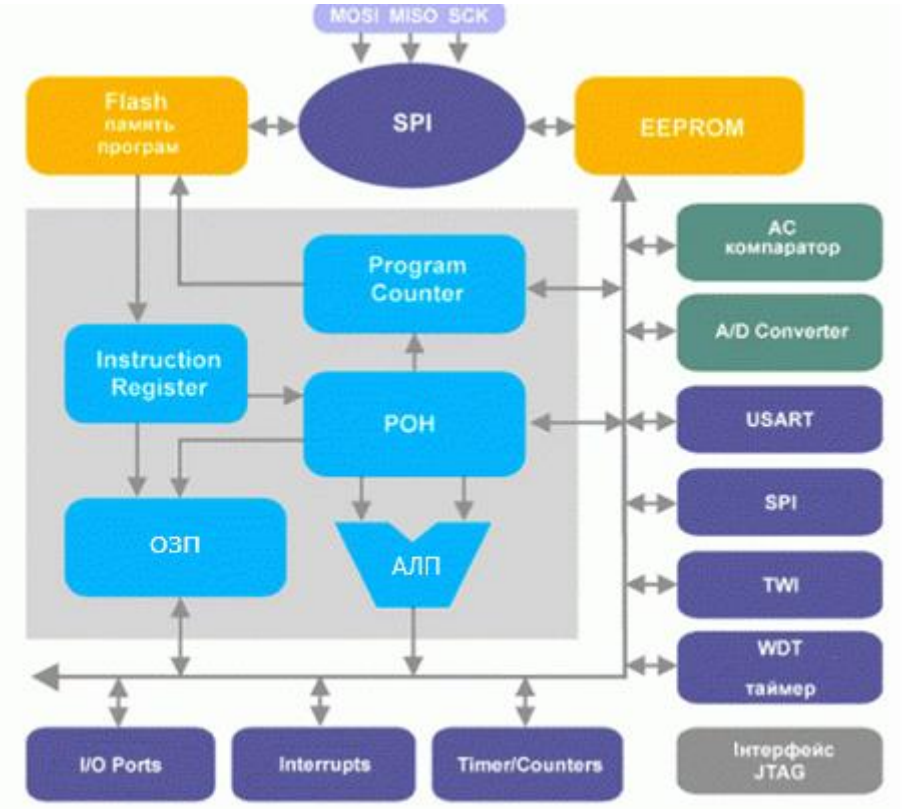


Рис. 4. Структура AVR-мікроконтролера

• АЛП - арифметико-логічний пристрій. Необхідний для виконання обчислення.

• Регістри загального призначення (РЗП) – регістри, які здатні отримувати дані й зберігати їх у той час, поки мікроконтролер підключений до живлення. Після перезавантаження дані стираються. РЗП слугують тимчасовими осередками для операцій з даними.

• Переривання - щось на зразок події, яка стається з внутрішніх або зовнішніх впливів на мікроконтролер - переповнення таймера, зовнішнє переривання з піна МК і т. д.

• JTAG - інтерфейс для внутрішньосхемного програмування без зняття мікроконтролера з плати.

• Flash, ОЗП, EEPROM - види пам'яті - програм, тимчасових робочих даних, довгострокового зберігання. Така пам’ять є незалежною від подачі живлення до мікроконтролеру відповідно до порядку в назвах.

• Таймери та лічильники - найважливіші вузли в мікроконтролері, у деяких моделях їх кількість може налічувати до десятка. Вони необхідні для того, аби вичитувати кількість тактів, відповідно часові відрізки, а лічильники збільшують своє значення з певної події. Їхня робота і режим залежать від програми, проте виконуються ці дії апаратно, тобто паралельно основному тексту програми, можуть викликати переривання (по переповнення таймера, як варіант) на будь-якому етапі виконання коду, на будь-якому його рядку.

• A / D (Analog / Digital) - АЦП, його призначення ми вже описали раніше.

• WatchDogTime (Сторожовий таймер) - незалежний від мікроконтролера і навіть його тактового генератора RC-генератор, який відраховує певний проміжок часу і формує сигнал скидання мікроконтролера, якщо той працював, і пробудження - якщо той був в режимі сну (енергозбереження). Його роботу можна заборонити, встановивши біт WDTE в 0.

Виходи мікроконтролера є досить слабкими, тобто, струм, що проходить через них, зазвичай не перевищує 20-40 міліампер, цього вистачить

для розпалювання світлодіоду і LED-індикаторів. Для більш потужного навантаження необхідні підсилювачі струму або напруги, наприклад, ті ж транзистори.

Мікроконтролери серії AVR мають розвинену систему команд. Кількість команд варіюється в залежності від моделі та може бути в межах від 90 до 133. Більшість команд займає лише одну комірку пам'яті і виконується за один такт. Це означає, що команди виконуються дуже швидко. Тобто швидкодія цих мікроконтролерів близька до максимально можливої.

Типовим представником 8-бітних однокристальних мікроконтролерів є КР580ИК80А (далі для стислості - КР580).

Даний мікроконтролер, виконаний за NMOn-технологією, містить приблизно 5000 елементів і реалізований в 40-вивідному корпусі. Число базових команд МП КР580 становить 78, час виконання команд для тактової частоти 2 МГц лежить в діапазоні 2-9 мкс. Команди мікроконтролера можуть бути одно-, двох- і трьохбайтними. Двох- і трьохбайтні команди зберігаються в сусідніх комірках пам'яті. Мікроконтролер КР580 має чотири режими адресації:

1. Пряма адресація. У цьому режимі другий і третій байти команди містять виконавчу адресу команди, причому в другому байті – молодші розряди, а в третьому - старші.
2. Реєстрова адресація - для звернення до внутрішніх регістрів мікроконтролера.
3. Безпосередня адресація, при якій в команді вказується 8- або 16-бітний операнд.
4. Непряма регістрова адресація

Структурна схема мікроконтролера КР580 приведена на рис. 5.

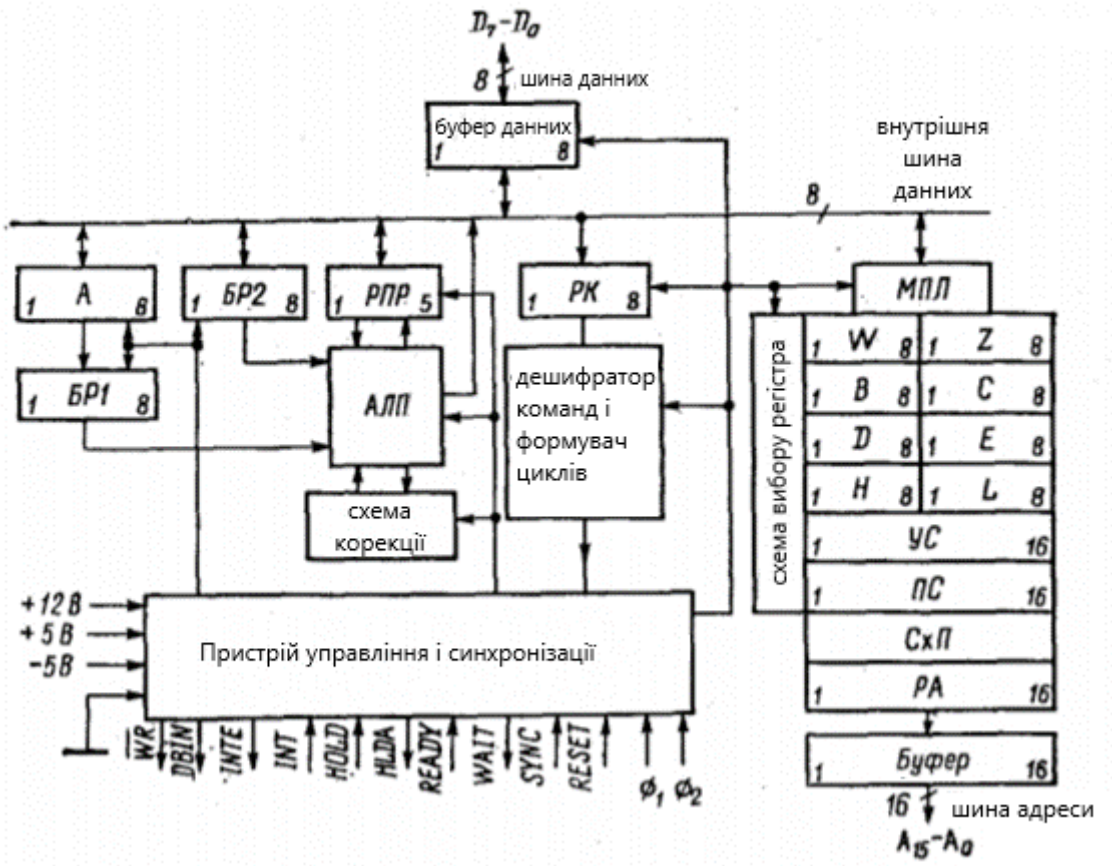


Рис. 5. Структурна схема мікроконтролера КР580

Функціональне призначення висновків мікроконтролера наступне:

A15-AT - шина адреси з трьома станами, що забезпечує адресацію пам'яті, ємністю до 64К байт, адресацію 256 портів введення і 256 портів виведення інформації;

D7-Do - двонаправлена шина даних з трьома станами, забезпечує обмін інформацією між мікроконтролерами, пам'яттю і периферійними пристроями;

DBIN - вихідний сигнал прийому, вказує пам'яті і периферійним пристроям, що шина даних знаходиться в режимі прийому інформації в мікроконтролері;

WR - вихідний сигнал видачі, використовують для управління видачею інформації з мікроконтролера до пам'яті й периферійних пристроїв, активним є сигнал WR = 0;

INT - вхідний сигнал переривання, що сприймається мікроконтролером після виконання поточної команди або в режимі зупинки; переривання не сприймається мікроконтролером, якщо він знаходиться в режимі захоплення або в режимі заборонених переривань, що забезпечується установкою тригера дозволу переривань в стан 0 командою DI;

INTE - вихідний сигнал дозволу переривань, відображає стан тригера дозволу переривання; даний тригер скидається в 0 після команди DI, а також після прийому сигналу переривання INT або сигналу скидання RESET;

HOLD - вхідний сигнал захоплення, переводить мікроконтролер у стан захоплення після завершення обміну даними між мікроконтролером, пам'яттю або периферійними пристроями в поточному машинному циклі; після захоплення шини даних і адреси переходять у стан високого вихідного опору;

HLDA - вихідний сигнал підтвердження стану захоплення;

READY - вхідний сигнал готовності, інформує мікроконтролер, що дані з зовнішнього джерела передані на шину даних, синхронізує роботу мікроконтролера з більш повільнодіючою пам'яттю або периферійними пристроями; при нульовому значенні цього сигналу мікроконтролер переходить в стан очікування Tw;

WAIT - вихідний сигнал очікування, що підтверджує, що мікроконтролер знаходиться в стані очікування Tw;

RESET - вхідний сигнал скидання, забезпечує установку в стан 0 регістра команд, програмного лічильника, тригерів дозволу переривання і підтвердження захоплення; при цьому стан інших регістрів не змінюється;

SYNC - вихідний сигнал синхронізації, визначає початок машинного циклу; 01, 02-тактові сигнали.

Системи команд мікроконтролера КР580 можемо поділити на такі 5 груп:

1. Команди пересилання кодів, що забезпечують пересилання даних між регістрами або пам'яттю і регістрами. Команди цієї групи не формують ознак результатів операцій.

2. Арифметичні команди, що забезпечують виконання операцій додавання і віднімання, зміни кодів на 1. Один операнд для бінарних операцій зберігається в акумуляторі А, інший - в регістрі або комірці пам'яті, а результат поміщається в акумулятор. Такі операції як множення і ділення виконуються програмним шляхом з використанням підпрограм. Негативні числа при виконанні арифметичних операцій необхідно перетворювати в додатковий код.

3. Логічні команди реалізують операції логічного додавання і множення, що виключає АБО, інвертування, ліве і праве зрушення і т. д. Вихідні операнди зберігаються в регістрах або елементах пам'яті, а результат поміщається в акумулятор.

4. Команди передачі управління, в число яких входять команди безумовної та умовної передачі управління, звернення і виходу з підпрограм. Дані команди не формують ознак результатів операцій.

5. Команди введення і виведення інформації, звернення до стекової пам'яті. Крім того, в цю групу входить ряд команд управління роботою мікроконтролера.

Контролер периферійного інтерфейсу (PIC) - це серія мікроконтролерів, розроблена компанією Microchip. Мікроконтролер PIC швидше і простіше реалізує програми у порівнянні з деякими іншими (наприклад, 8051). Простота програмування і простота взаємодії з іншими периферійними пристроями робить PIC більш успішним мікроконтролером.

Мікроконтролери PIC мають RISC-архітектуру. RISC - скорочений набір команд, використовується також в процесорах для мобільних пристроїв.

Серед 8-бітних мікроконтролерів PIC налічує 3 родини, які відрізняються архітектурою (розрядністю і набором команд).

1. Baseline (PIC10F2xx, PIC12F5xx, PIC16F5x, PIC16F5xx);
2. Mid-range (PIC10F3xx, PIC12F6xx, PIC12F7xx, PIC16F6xx, PIC16F7xx, PIC16F8xx, PIC16F9xx);
3. Enhanced Mid-range (PIC12F1xxx, PIC16F1xxx);
4. High-end або PIC18 (18Fxxxx, 18FxxJxx and 18FxxKxx).

Мікроконтролери цієї серії мають скорочену систему команд. Тобто, якщо звичайні мікроконтролери мають кілька сотень команд, то мікроконтролери серії PIC – кілька десятків.

Мікроконтролери PIC програмуються з використанням послідовного протоколу. Тому кінцевому користувачу недостатньо підключити мікросхему контролера PIC безпосередньо до будь-якого «стандартного» інтерфейсу.

Однак технічні вимоги до програмування за часом досить слабкі. Цей фактор дає можливість використовувати деякі висновки паралельного або послідовного порту комп'ютера для генерації програмної послідовності за допомогою програмного забезпечення.

Крім робочої напруги, мікроконтролера необхідні три види сигналу:

1. Запрограмована напруга (близько 13В).
2. Таймер програмування (ICSPCLK).
3. Дані (ICSPDAT).

Оскільки більшість зразків PIC допускають запрограмовану напругу трохи нижче значення, вказаного специфікацією, з’являється можливість використання рівнів сигналів ± 12В, наявних на інтерфейсі послідовного порту настільного ПК. Таким чином, «записати» PIC цілком допустимо без необхідності підключення додаткового джерела живлення.

На рисунку 6 наведено структуру мікроконтролера PIC.

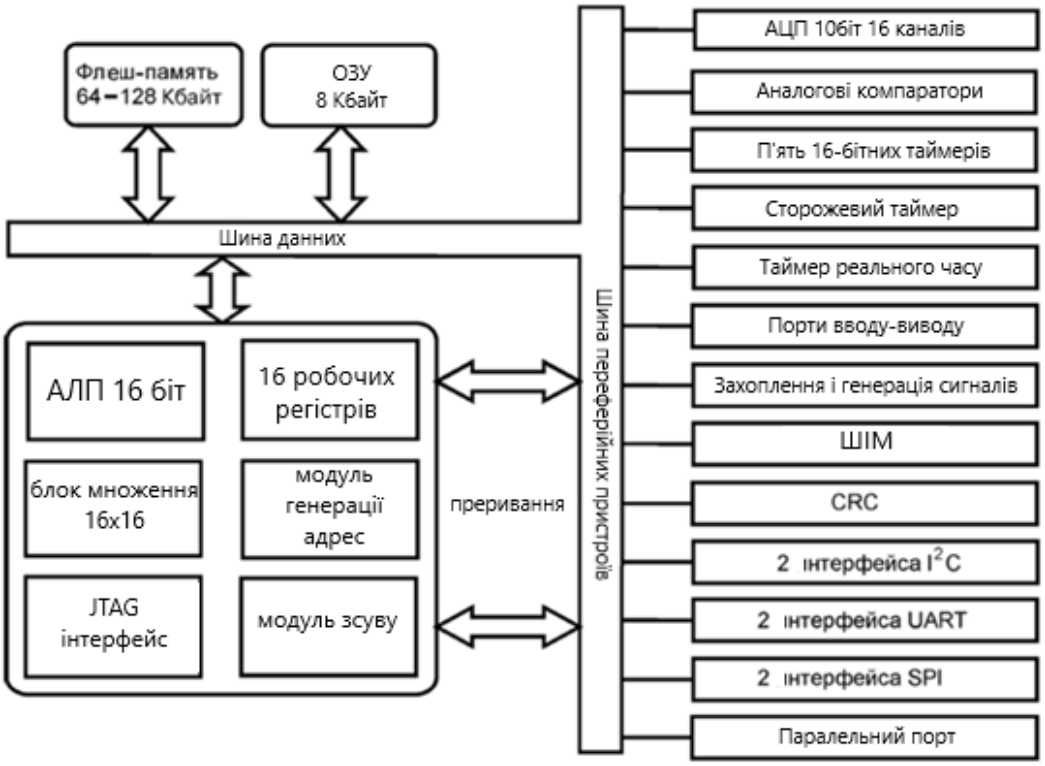


Рис. 6. Загальна структура мікроконтролера PIC

Мікроконтролери STM32 виконані на базі ядра ARM Cortex. На перший погляд, різниця між мікроконтролерами STM32 і мікроконтролерами іншої фірми непомітна, набір вбудованих ПВВ (пристроїв введення-виведення) такі як АЦП, таймери загального призначення, I2C, SPI, CAN, USB і годинник реального часу мають і звичайні мікроконтролери. Але якщо розглядати кожне з цих ПВВ більш детально, стане очевидно, що в STM32 вони влаштовані набагато складніше. Наприклад, АЦП – аналогово-цифровий перетворювач - має розрядність в 12 біт, а також вбудований датчик температури, і підтримує кілька режимів перетворення вхідних даних. Таймери оснащені блоками захоплення і порівняння, так як можуть бути використані як окремо, так і синхронно, що дозволяє створювати великі масиви таймерів. У STM32, як було сказано раніше, є, так звані, розширені таймери (advanced timers), їх використовують для управління електродвигунами. Для цього у них передбачено шість комплементарних ШІМ-виводів з програмованим мертвим часом (dead-time).

На відміну від інших мікроконтролерів, в STM32 передбачений модуль DMA - прямий доступ до пам'яті, кожен канал даного модуля може бути використаний для передачі даних між регістрами будь-якого з ПВВ і запам’ятовуючими пристроями.

Ще одним плюсом мікроконтролерів STM32 є поєднання характеристик малого енергоспоживання і досить високої продуктивності. Вони здатні працювати лише від 2В-ого джерела живлення на тактовій частоті 72Мгц і споживати в активному стані всього лише 36мА, якщо ж використовувати підтримувані ядром Cortex економні режими роботи, то можна знизити енергоспоживання до незначних 2мА.

За безпеку в даному типі мікроконтролерів відповідають два сторожевих таймери (watchdog), які дозволяють в разі помилки виконання програми перезавантажити мікроконтролер автоматично і продовжити виконання.

Шина ЛРВ2 обслуговує такі периферійні пристрої:

- порти I / O загального призначення (General purpose I / O, GPIO);
- альтернативні функції портів I / O (Alternate function I / O, AFIO);
- контролери послідовних інтерфейсів (USART1 і SPI1);
- контролер аналого-цифрових перетворювачів (ADC 1 і ADC2);
- таймер-лічильник (ТС 1);
- зовнішні переривання (EXTI).

Шина АРВ1 обслуговує такі периферійні пристрої:

- контролери інтерфейсів (USART2,3, SPI2,12C1,2);
- таймери-лічильники (ТС2, 3,4);
-сторожеві таймери -1WDT (незалежний) і WWDT (віконний);
- контролери послідовних інтерфейсів (USB і CAN);
- контролер управління електроживленням (PWR);
- регістри для резервного копіювання даних (ВКР).

Як CPU Cortex, так і блок ПДП можуть бути використані як шинні майстри. Завдяки властивому матриці шин паралелізму, необхідність в арбітражі виникає тільки в разі спроб одночасного доступу обох майстрів до статичного ОЗП, шини APB1 або APB2. Проте шинний арбітр гарантовано надає 2/3 часу доступу для блоку ПДП і 1/3 для CPU Cortex. У структурі внутрішніх шин передбачені окрема шина інструкцій і матриця шин, яка надає кілька каналів передачі даних для CPU Cortex і блоку ПДП, що видно на рисунку 7.

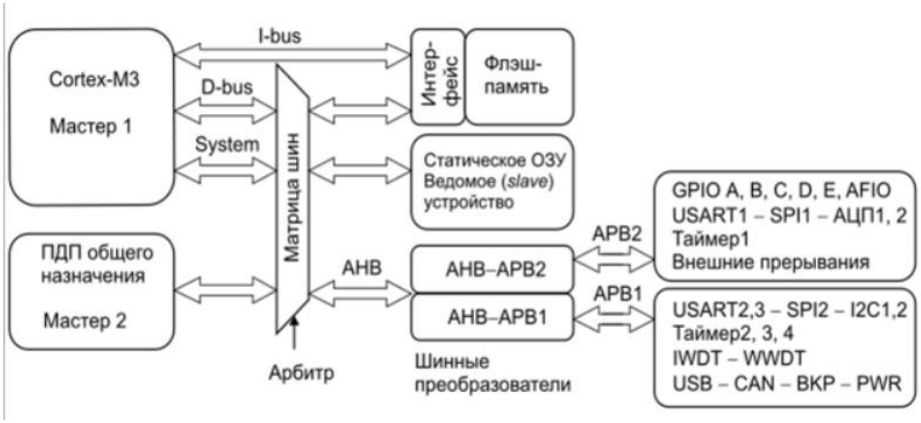


Рис. 7. Організація внутрішніх шин STM32

1.3. Мікроконтролери AVR та STM, їхні параметри, переваги, порівняльний аналіз

Ключові параметри, необхідні для ґрунтовного зіставлення мікроконтролерів AVR та STM, наведено нижче:

Таблиця 1 – Порівняння мікроконтролерів AVR та STM

Характеристика	STM32F103C8T6	AVR (Arduino Nano)
Частота контролера, МГц	72	16
Пам’ять програм, кБайт	64	32
Живлення, В	3,3	5
ОЗП, кБайт	20	2
USB 2.0	+	-
DMA	+	-
CAN	+	-
RTC	+	-
UART	3	1
Прошивка через USB	-	+
Ціна	2.1$	1.8$

За критерієм апаратного аспекту плата STM32 повністю затьмарює Arduino Nano. Мікроконтролер STM32 працює на тактовій частоті 72 МГц, в той час як Arduino - лише на 16 МГц. Обсяг Flash-пам'яті більше у два рази, а обсяг оперативної пам'яті - в десять разів. Крім того, у STM32 три канали UART, по два канали SPI і I2C, а у Arduino тільки по одному каналу. Крім цього, STM32 може підключатися до шини CAN. При всіх цих характеристиках плата STM32 має майже ідентичний форм-фактор в

порівнянні з Arduino Nano. Плюсом STM32 є ще і вбудовані контролер DMA і годинник реального часу. Також всі лінії STM32 є толерантними до напруги 5 В, тому не потрібні ніякі перетворювачі рівня. Також є можливість запрограмувати плату STM32 за допомогою Arduino IDE, завантаживши версію з підтримкою STM32. STM32 має у своєму арсеналі RTC та контролер DMA (прямий контролер доступу до пам'яті).

Сімейство мікроконтролерів STM32 відрізняється від своїх конкурентів відмінним поведінкою при температурах від -40°C до +80°C. Висока продуктивність не зменшується, на відміну від Arduino. Також можна знайти вироби, що працюють при температурах до 105°C.

Отже, можемо говорити про такі переваги мікроконтролерів STM32 у порівнянні з AVR як:

1. Низька вартість;
2. Зручність використання;
3. Великий вибір середовищ розробки;
4. Взаємозамінність чіпів - якщо не вистачає ресурсів одного мікроконтролера, його можна замінити на більш потужний, не змінюючи самої схеми і плати;
5. Висока продуктивність;
6. Зручна відладка мікроконтролера.

Проте варто вказати і на недоліки, а саме: високий поріг входження; відсутність актуальної сучасної літератури з STM32; застарілість більшості створених бібліотек, а відтак і нагальна потреба у створенні власних.

Рис. 8 Arduino Nano (праворуч) та STM32F103C8T6 (ліворуч)

Саме через вищеназвані ознаки мікроконтролери серії STM32 були використані у процесі створення автоматизованого робочого місця, а саме: вже згаданий у порівняльному аналізі STM32F103C8T6 (рис. 8) та STM32F407VGT6 (рис. 9).

Рис. 9. Мікроконтролер STM32F407VGT6

Мікроконтролер STM32F103C8T6 має 64Кб Flash-пам'яті і тактову частоту ядра 72МГц. Виконаний на базі процесора ARM Cortex M3 і має 32- хбітну архітектуру. Має в наявності один розширений таймер TIM1, за допомогою якого можна генерувати трьохфазний ШІМ-сигнал.

Мікроконтролер STM32F404VGT6 має 1Мб Flash-пам'яті і тактову частоту ядра 192Мгц. На відміну від попереднього, STM32F4VGT6 виконаний на базі процесора ARM Cortex M4, який так само має 32-х бітну архітектуру. В наявності є вже два розширених таймери, а також двоканальний ЦАП, який буде використовуватися в дослідженнях.

1.4. Розробка схеми пристроїв на мікроконтролерах

Сімейство мікроконтролерів STM32 побудовано з використанням 32-розрядного ядра Cortex різних версій (в мікроконтролері, встановленому на платі використовується ядро Cortex-M4). Деякі основні характеристики ядра мікроконтролерів STM32 представлені в таблиці 2.

Таблиця 2. Основні характеристики ядра мікроконтролерів STM32

Характеристика	Значення
Ширина слів для даних, розряд	32
Архітектура	Гарвард
Конвеєр	3-ступеневий
Набір інструкцій	RISC
Буфер, розряд	2x64
Середній розмір інструкції, байт	2
Тип переривання	Векторизований
Затримка реакції на переривання	12 циклів
Режим керування енергоспоживанням	Сон, сон по виході, глибокий сон
Налагоджувальний інтерфейс	ST-LINK, JTAG

Мікроконтролери даного типу побудовані на гарвардській архітектурі і мають 3-ступеневий конвеєр, який мінімізує час виконання команд. Вони розроблені для побудови систем з максимальною енергоефективністю і мають кілька режимів управління енергоспоживанням. У них використовуються внутрішні інтерфейси пам'яті, шириною більше, ніж середня довжина інструкції. Це мінімізує число доступів до шини пам'яті, а, отже, і споживання електроенергії, пов'язане з операціями по шині і зчитуванням незалежної пам'яті. Технологія безперервної обробки переривань з винятком внутрішніх операцій над стеком (tail chaining) скорочує час реакції на переривання і виключає зайві операції зі стеком.

На рис. 10 представлено спрощене уявлення цифрового периферійного пристрою.

Периферійний вузол може бути розділений на два основні блоки. Перший блок - це ядро, яке містить кінцеві автомати, лічильники і будь-який вид комбінаторної або послідовної логіки. Воно призначене для виконання завдань, які потребують участі процесора, таких як прості завдання передачі даних, управління аналоговими входами або виконання функцій, прив'язаних до синхросигналів. Ядро периферійного вузла зв'язується із зовнішнім світом через порти введення / виводу мікроконтролера. Зовнішні з'єднання можуть складатися з декількох сигналів або складних шин. Другий блок - налаштування і управління периферією, які здійснюються додатком через регістри, з'єднані з внутрішньою шиною, що розділяється з іншими ресурсами мікроконтролера.

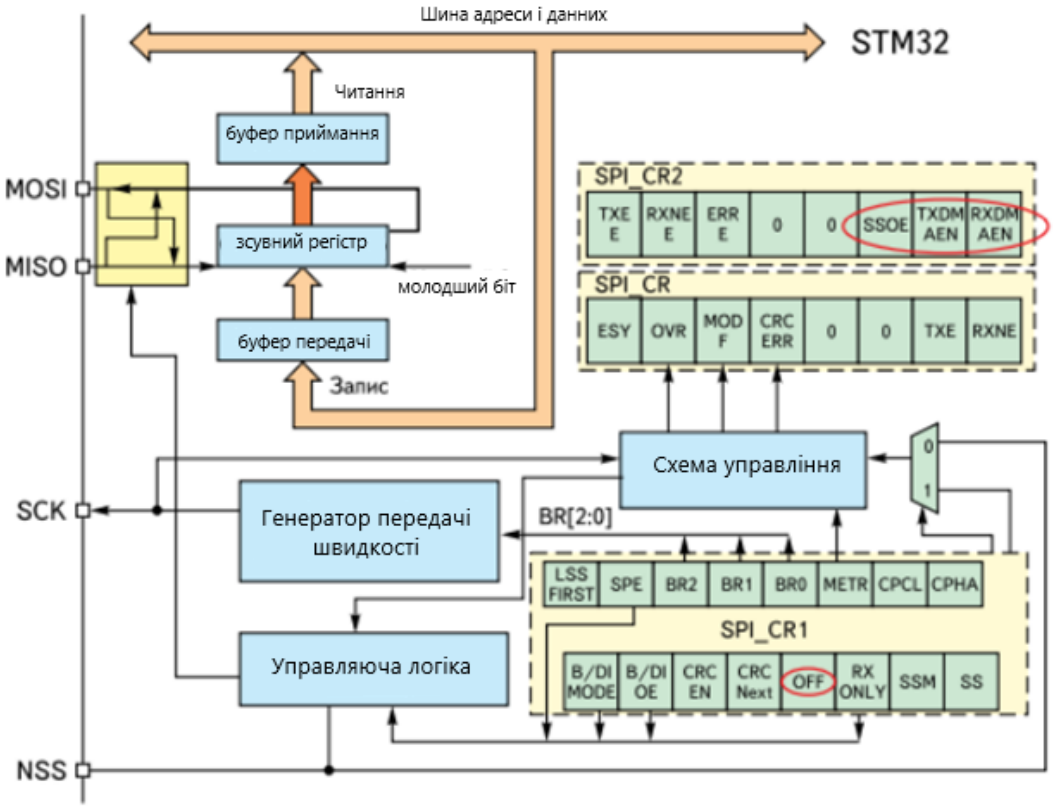


Рис. 10. Подання цифрового периферійного пристрою

РОЗДІЛ ІІ. АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОКОНТРОЛЕРІВ STM32

2.1. Програмування та прошивка мікроконтролерів STM32

Програмування мікроконтролерів STM32 зводиться до написання алгоритму програми на мові програмування і запису виконуваної програми до пам'яті контролера.

Для написання програм буде використовуватися мова C(Ci). C(Ci) – компільована, статично типізована мова програмування загального призначення, розроблена в 1969-1973 роках працівником Bell Labs Деннісом Рітчі.

Спочатку код пишуть зрозумілою для людини мовою в середовищі програмування IDE, потім, за допомогою компілятора, переводять в так званий машинний код і записують за допомогою програматора до пам'яті мікроконтролера.

Для програмування STM32 мовою C існує безліч платних і безкоштовних розробок. Також існують компілятори для різних мов програмування. Але в даній розробці використовується мова програмування Ci, отже, і компілятор буде використаний для цієї мови.

Для того, щоб навчитися розробляти програмне забезпечення для мікроконтролерів, які будуть використані в лабораторних роботах, необхідно навчитися спілкуватися з ними однією мовою.

Коментарі необхідні для пояснення коду програми і не є частиною коду. Коментарі бувають багаторядкові і однорядкові. Однорядкові і багаторядкові коментарі докладніше розглянуті в таблиці 3.

Таблиця 3. Види коментарів

Коментарі	
// Однорядковий коментар	Якщо помістити // на будь-який рядок, то компілятор проігнорує весь текст після цього символу на цьому рядку. Зазвичай використовується для пояснення визначеного рядка коду.
/ * Багаторядковий коментар * /	Багаторядкові коментарі починаються з / * І закінчуються * /. Багаторядкові коментарі необхідні при розкритті цілого блоку коду, наприклад, опису програми.

Всі константи і оператори розташовуються у FLASH-пам'яті (ПЗП, RAM, пам'ять програм), вміст даної пам'яті залишається незмінним у процесі роботи програми. Для зберігання даних, які можуть бути виконані в процесі виконання програми, використовується оперативна пам'ять (ОЗП, RAM, пам'ять даних), в цій пам'яті так само розташовуються всі змінні. Змінна – пойменована адреса, або така, що адресується іншим способом області пам'яті, яку можна використовувати для здійснення доступу до даних і змінювати значення в ході виконання програми. Для введення змінної в мові С використовують наступні ключові слова:

1. char - створює змінну розміром 1 байт (8 біт), яка може приймати 256 значень.
2. int - дійсний тип, розмір створюваної змінної буде залежати від архітектури використовуваного мікроконтролера, для STM32 розмір

буде дорівнювати 32 біти, і буде займати в пам'яті мікроконтролера 4 байти.

3. short - дійсний тип, розмір змінної 16 біт, це означає, що змінна може приймати 65536 значень і займає в пам'яті 2 байти.
4. long - дійсний тип, розмір змінної 32 біти, це означає, що змінна може приймати 232 значень і займає в пам'яті 4 байти.
5. float - змінна використовується для операцій з числами з плаваючою комою, займає 4 байти. На дію з даним типом змінної необхідно затратити більше тактів процесора, тобто більше часу.

Для числових змінних існують модифікатори, які указують на наявність знаку в змінної.

1. unsigned - вказує на те, що змінна не матиме знаку, тобто буде приймати тільки позитивні значення, наприклад unsigned char створить змінну з діапазоном значень від 0 до 255.
2. signed - вказує на те, що змінна буде мати знак, наприклад, *signed char* створить змінну з діапазоном значень від -128 до +128.

Для зберігання строкових даних, а також для зберігання великої кількості однотипних значень, зручно використовувати масиви. Масив, як і змінна, має тип, а також своє унікальне ім'я. На відміну від змінної, при визначенні масиву необхідно вказати кількість вхідних до нього елементів. Надалі до цілого масиву можна звертатися, просто написавши його ім'я, а до певного елементу масиву можна звернутися за індексом елемента. Індекс елемента в масиві - це порядковий номер елемента, починаючи з нуля.

Імена змінних, функцій і констант в С повинні обиратися, виходячи з таких правил:

1. Імена повинні починатися з букв латинського алфавіту (a - z, A - Z), або зі знака нижнього підкреслення «_».
2. В іменах змінних, констант, а також функцій дозволено використовувати тільки букви латинського алфавіту, цифри (0-9), а

також символ нижнього підкреслення «_», інші символи використовувати заборонено.

3. У мові С регістр букв має значення, це означає, що застосовувані літери нижнього регістру (a - z) і літери верхнього регістру (A - Z) відрізняються. Наприклад, імена змінних: name, Name, NaMe - різні.
4. Імена змінних, констант, функцій можуть мати довжину у відповідності до стандарту ANSIC, що не перевищує 32 символи.
5. Ідентифікатори для нових об'єктів не повинні збігатися з ключовими словами мови, а також з назвами бібліотечних функцій.

Якщо необхідно записати значення змінної в шістнадцятковому вигляді, то використовується знак «0x» (наприклад, 0xAAA), якщо необхідно створити змінну в двійковому вигляді, то використовується знак «0b» (наприклад, 0b1001).

Різні види змінних представлені в таблиці 4.

Таблиця 4. Деякі види змінних

Введена змінна	Область застосування змінної
unsigned char name_value1;	Дана змінна може використовуватися, коли заздалегідь відомо, що значення буде змінюватися, наприклад, у діапазоні від 0 до 100, не перевищує розмірність 1 байт, а також має тільки позитивні значення.

unsigned short name_value2;	Дана змінна може використовуватися тоді, коли заздалегідь відомо, що значення буде змінюватися, наприклад, в діапазоні від 50 до 5000, не перевищує розмірність 2 байт, а також має лише позитивні значення
signed int name_value3;	Дана змінна підходить для діапазону значень, наприклад, від - 30 до +6000, має розмірність 4 байти
unsigned int name_array [100];	Даний масив являє собою набір зі ста змінних типу unsigned int з іменем name_array.

Діапазон можливих значень для різних типів змінних представлений в таблиці 5.

Таблиця 5. Діапазон значень для різних змінних

Тип змінної	Діапазон значень для unsigned	Діапазон значень для signed
char	Від 0 до 255	Від -128 до +127
short	Від 0 до 65535	Від -32768 до +32767
int (для Cortex-M3)	Від 0 до 4294967295	Від - 2147483648 до +2147483647
long	Від 0 до 4294967295	Від - 2147483648 до +2147483647

long long	від 0 до 18446744073709551615	від -223372036854775808 до +22337203685477580 7

Для здійснення операцій над змінними існують оператори дії:

1. «+» - оператор математичного складання, наприклад, a + b, складає дві змінні a і b.
2. «-» - оператор математичного вирахування, наприклад, ab.
3. «*» - оператор математичного множення, наприклад, a*b, перемножування двох змінних a і b.
4. «/» - оператор математичного розподілу, наприклад, a / b.
5. «%» - оператор залишку від ділення, визначає залишок від ділення, наприклад, a% b, 6/4 = 1.5, залишок від ділення 0.5, отже, при 6% 4 = 0.5.
6. «=» - оператор присвоювання значення, наприклад, a = b, привласнює змінній a значення змінної b.
7. «<» - операція порівняння - менше.
8. «>» - операція порівняння - більше.
9. «<=» - операція порівняння - менше або дорівнює.
10. «> =» - операція порівняння - більше або дорівнює.
11. «==» - математична операція порівняння - дорівнює.
12. «! =» - математична операція порівняння - не дорівнює.
13. «++» - інкремент, наприклад, a ++, інкремент числа a, збільшення на одиницю.
14. «-» - декремент, наприклад, a--, декремент числа a, зменшення на одиницю.

15. «<<» - операція побітового зрушення вліво, наприклад, a << b, зрушує значення a вліво, на кількість біт b, 23 << 1, число 23 в двійковій системі числення має вигляд 00010111, зсувається на 1 вліво, виходить 0 0010 1110, що відповідає числу 46 в десятковій системі числення.

16. «>>» - операція побітового зрушення вправо, наприклад, a >> b, зрушує значення a вправо, на кількість біт b, 23 >> 1, число 23 в двійковій системі числення має вигляд 00010111, зсувається на 1 вправо, виходить 0000 1011 , що відповідає числу 11 в десятковій системі числення.

17. «&» - операція порозрядове логічне «І», наприклад, a & b. Також дана операція називається логічним множенням.

18. «|» - операція порозрядове логічне «АБО», наприклад, a | b. Також дана операція називається логічним складанням.

19. «~» - операція логічне «НЕ», наприклад, ~ a, інвертує значення змінної a, тобто логічні нулі всіх розрядів стають логічними одиницями, а логічні одиниці стають нулями.

Мова програмування не є повноцінною без використання логічних операторів. У мові С існує чотири основних оператора: if, while, for, switch.

Оператор if виробляє умовний перехід до виконання різних частин програм. Оператор має певну структуру (рисунок 11).

```
if(умова)
{
Код виконується при виконанні"умови"
}
else
{
Даний код виконується якщо "умова" не виконана
}
```

Рис. 11. Структура оператора if

У круглих дужках після if вказується умова, при застосуванні якої буде виконано код, який укладено між першими фігурними дужками. Якщо умова не виконується, то буде виконуватися код, який укладено між фігурними дужками після else. Наявність else необов'язкова, якщо не потрібне виконання коду в разі не виконання умови, то else можна не писати.

Умовою дуже часто вказують вираження порівняння, наприклад, a == b або a! = B. Фігурні дужки означають початок і кінець коду. Якщо необхідно використовувати відразу декілька умов, то такі умови об'єднують операторами логічних зв'язків «&&» - логічне «І», «||» - логічне «АБО».

Для створення циклів, що дають можливість багаторазово повторювати один і той же код до досягнення якогось умови, існують цикли while і for.

Синтаксис циклу while дуже простий, в круглих дужках, які слідують відразу ж за словом while, вказується умова, поки дана умова буде виконуватися, буде виконуватися і код, укладений у фігурні дужки після while (рисунок 12).

```
while(умова)
{
Код виконується поки виконується умова
}
```

Рис.12. Структура циклу while

Цикл while необхідно застосовувати з обережністю, оскільки можливо створити ситуацію, коли умови циклу не будуть виконані ніколи, це призведе до зависання програми. Безпечніше використовувати цикл for. Даний цикл призначений для виконання заздалегідь заданої кількості повторень коду (рисунок 13). Синтаксис написання циклу for наступний: for (завдання початкової умови; умова; операція зі зміни умови).

```
unsigned char i;
    //Змінна для умови в циклі for
    for(i=0;i<10; i++)
    {
    a++;
    //Данний код буде виконуватись поки і не буде дрівнювати 10
    }
```

Рис.13. Структура циклу for

Дуже часто програма повинна виконувати одну і ту ж послідовність дій, при достатньому повторі таких дій і з метою поліпшення читаності коду, ці послідовності створюють у вигляді окремої групи. Коли необхідно знову виконати дану послідовність, програму просто направляють до даного блоку. Такий окремий блок коду в мові С називаюьб функцією.

Основними передумовами для створення функцій є:

– наявність однакових, часто повторюваних дій;

– бажання структурувати програму у вигляді окремих блоків для зручності прочитання коду.

Використання функцій має істотний мінус, наприклад, при переході до функції і поверненні з неї мікроконтролер змушений зберігати деякі дані, наприклад, адресу програми, а це неминуче позначиться на швидкості розрахунку.

Будь-яка функція має свій унікальний заголовок, список переданих їй змінних, тип змінної, що повертається, і, найголовніше, це тіло функції, з вмістом її коду. Ім'я функції обирається за тими ж правилами, що й ім'я для змінної. Загальний вигляд функції представлений на рисунку 14.

```
повертається_тип_змінної function_name(переданий_тип_змінної)
{

}
```

Рис.14. Загальний вигляд функції

Якщо функція не повинна приймати і / або повертати дані, то в якості типу даних необхідно вказати void. Будь-яка функція може повертати тільки одну змінну, але приймати може безліч змінних, вказаних в дужках через кому. Приклад функції наведено на рисунку 15.

```
void delay(unsigned int n)
{
        while(n)
        {
                n--;
        }
}
```

Рис.15. Приклад функції з void

Для того, щоб мати можливість викликати функції, розташовані як вище, так і нижче місця виклику, на самому початку файлу необхідно прописати імена функцій (оголосити функції) із зазначенням даних, що ними передаються і повертаються.

У великих програмах з'являється безліч різних функцій, для зручності виклику цих функцій їхні прототипи виносять в окремі файли, які називаються заголовками. Заголовковий файл має розширення «.h» і підключається до файлу вихідного коду (з розширення «.c») за допомогою директиви #include <>. Для зручності читання коду зазвичай створюють кілька файлів з вихідним кодом (розширення «.c») і відповідні їм заголовкові файли з розширенням «.h».

При використанні в різних частинах коду безлічі констант, що постійно повторюються, або інших невеликих фрагментів коду, таких як, наприклад, перевизначення висновків мікроконтролера, для спрощення вимірювання цих місць, а також для поліпшення читаності коду, використовують препроцесорну директиву #define. Загальний вигляд макросу: #define імя_макроса замінний рядок. Команда #define використовується для організації заміни по всьому файлу. Іншими словами, #define повідомляє компілятору, що необхідно замінити «Ім'я_макросу» на «замінний рядок».

Передпроцесорна директива #define може використовувати логічні умови, подібно «if ... else». Форма запису логічних умов представлена на рисунку 16.

```
#ifdef ADC //якщо макрос ADC був визначений раніше
//Дія виконується, якщо ADC був визначений раніше
#else //якщо ADC не був визначений раніше
//Дія виконується, якщо ADC був визначений раніше
#endif // Кінець умови
```

Рис.16. Логічні умови для макросів

Крім #ifdef , також використовуються й інші умови:

#ifndef - подібно до #ifdef, але виконується, якщо вказаний макрос раніше не був визначений;

#if - подібно до if порівнює виконання логічної умови для макросів.

Програмісти, що пишуть програми на мові С, які використовують макроси, зазвичай в якості ідентифікаторів використовують символи верхнього регістру. Якщо розробник буде слідувати цьому правилу, то той, хто буде читати його програму, відразу ж зрозуміє, де буде відбуватися макрозаміна.

Для того, щоб зрозуміти особливості програмування STM32, необхідно детально розібрати один приклад. Для прикладу буде обрана найпростіша програма, також відома як «Hello world! для мікроконтролерів» - запалити світлодіод, розташований на налагоджувальній платі. Для даного прикладу використовується налагоджувальна плата на базі мікроконтролера STM32F103C8T6 (рисунок 17).

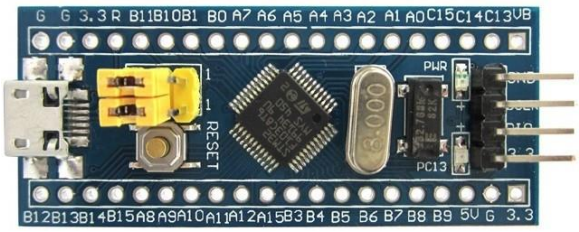


Рис.17. Налагоджувальна плата на базі STM32F103C8T6

Плата має 32 виведення для «спілкування» мікроконтролера із зовнішнім світом, які можуть бути налаштовані як на вхід, так і на вихід. Наявна також кнопка скидання, яка дозволяє припинити виконання програми і почати все спочатку. Також плата має інтерфейси Micro-USB і SWD для програматора. Дана налагоджувальна плата також має світлодіод, який підключено до порту С виводу 13.

Для завантаження робочої програми в мікроконтролер через інтерфейс SWD буде використовуватися програматор ST-LINK V2 (рисунок 18) для

мікроконтролерів STM32.

Рис.18. Програматор ST-LINK V2

Для вивчення основ програмування необхідно ввести деякі визначення:

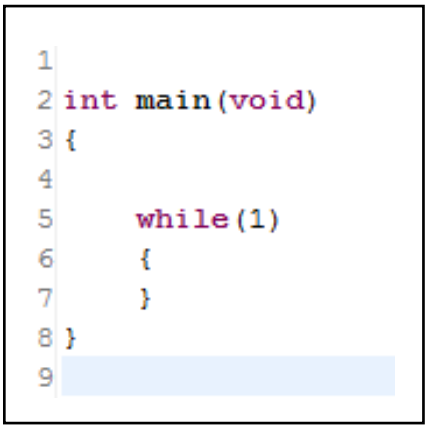
– бібліотека - збірник підпрограм, які використовуються для розроблення програмного забезпечення;

– структура - спеціальна конструкція, що дозволяє утримувати в собі безліч змінних різного типу;

– регістр процесора - блок осередків пам'яті, який утворює надшвидку оперативну пам'ять;

– переривання - сигнал від програмного або апаратного забезпечення, що повідомляє процесору про настання якої-небудь події, переривання бувають внутрішніми і зовнішніми.

Програма складається з функцій (підпрограм), їх може бути скільки завгодно, але вони повинні бути обов'язково написані в основній функції main (), так як дана функція є вхідною точкою, що знаходиться в цій функції. Загальний вигляд функції main () представлений на малюнку 19.



```
1
2 int main(void)
3 {
4
5     while(1)
6     {
7     }
8 }
9
```

Рис.19. Функція main ()

Між фігурними дужками розташовані блоки програм. У функції main представлено два блоки програм: перший від самої функції, в ньому будуть розташовуватися всі підпрограми розробленого коду; другий блок належить нескінченному циклу while.

Розробка будь-якого коду починається з вибору і підключення бібліотек. Для того, щоб запалити світлодіод, необхідно всього дві бібліотеки - RCC і GPIO.

Бібліотека RCC використовується для управління тактовим генератором мікроконтролера. Тактовий генератор мікроконтролера дозволяє подавати напругу на порти введення-виведення й іншу периферію, так як однією з особливостей мікроконтролерів STM32 є вимкнена периферія для економії енергії.

Бібліотека GPIO використовується для управління GPIO (General Ports Input- Output) портами введення-виведення. Дозволяє налаштовувати висновки мікроконтролера на вхід або на вихід, зчитувати стан висновків, отримувати вхідні дані і т. д.

Заголовковий файл - файл, вміст якого автоматично додається препроцесором до вихідного тексту. За традицією, в заголовкових файлах розміщують бібліотечні функції. Підключаються за допомогою директиви

#include <>, де між «<>» має бути записана назва заголовкового файлу. Приклад підключення заголовних файлів представлений на рисунку 20.

```
1 #include <stm32f10x_rcc.h>//бібліотека для тактового генератора
2 #include <stm32f10x_gpio.h>//бібліотека для GPIO
3 int main(void)
4 {
5
6     while(1)
7     {
8     }
9 }
```

Рис. 20. Приклад підключення заголовкових файлів

Коли підключення заголовкових файлів завершено, необхідно приступити до розробки самого коду. Спочатку необхідно включити тактування порту, на якому знаходиться світлодіод, на самій платі вказано, що це порт С. Тактування портів в даному мікроконтролері вмикається функцією RCC_APB2PeriphClockCmd (), APB2 - це шина до якої підключені всі порти введення-виведення даного мікроконтролера. Функція RCC_APB2PeriphClockCmd () має два аргументи, перший аргумент - це периферія, яку збираємося включити (наприклад, RCC_APB2Periph_GPIOC, так вмикається тактування порту С), другий аргумент - це включення або відключення тактування, і може приймати значення або ENABLE, або DISABLE відповідно.

Після включення тактування порту необхідно налаштувати вивід 13 як вихід. Для цього потрібно створити і заповнити структуру, поняття «структура» було введено раніше. Для створення структури портів введення-виведення використано ключове слово GPIO_InitTypeDef. Потім структура повинна бути заповнена полями для даного мікроконтролера, щоб визначити висновок, необхідно заповнити всього три поля:

1) GPIO_Mode – визначає, в якості чого буде налаштований вивід, може приймати значення:

а) GPIO_Mode_AIN - аналоговий вхід;

б) GPIO_Mode_IN_FLOATING - диференційний вхід;

в) GPIO_Mode_IPD - вхід з підтягуючим резистором до землі;

г) GPIO_Mode_IPU - вхід з підтягуючим резистором до живлення;

д) GPIO_Mode_Out_OD - вихід з відкритим стоком;

е) GPIO_Mode_Out_PP - вихід двома станами;

ж) GPIO_Mode_AF_OD - вихід з відкритим стоком для альтернативних функцій. Використовується у випадках, коли виводом повинна керувати периферія, прикріплена до такого виводу порту (наприклад, висновок Tx USART1 і т. п.);

з) GPIO_Mode_AF_PP - вихід для альтернативних функцій з двома станами;

2) GPIO_Pin - визначає номер виводу;

3) GPIO_Speed - визначає швидкість тактування виведення.

Для включення світлодіоду потрібно налаштувати вивід як вихід, для цього необхідно в полі GPIO_Mode вказати значення GPIO_Mode_Out_PP. В полі GPIO_Pin необхідно вказати значення GPIO_Pin_13, так як світлодіод знаходиться на 13-му виводі. У полі *GPIO_Speed* необхідно вказати значення *GPIO_Speed_2MHz,* такої частоти достатньо для того, щоб запалити світлодіод. Після заповнення всіх полів необхідно ініціалізувати структуру за допомогою функції GPIO_Init (), ця функція передає значення полів структури до порту GPIO. Функція GPIO_Init () має два аргументи, перший аргумент - найменування порту, в який буде надсилатися структура, в даному випадку, це GPIOC, другий аргумент - це найменування структури, перед якою обов'язково ставиться знак «&». Знак «&» називається вказівкою на адресу пам’яті.

2.2 Опис середовища програмування

Для даного проекту було використано середовище програмування CooCox CoIDE. Це вільне інтегроване середовище розробки, орієнтоване на мікроконтролери на базі ARM Cortex. По суті, CoIDE є текстовим редактором для коду, компілятор для мови Ci (GCC for ARM) необхідно встановлювати і підключати окремо. Основною перевагою даного середовища розробки є те, що завдяки спеціальному сховищу, при під'єднаному інтернеті користувач має доступ до безлічі бібліотек і прикладів. Загальний вигляд вікна середовища програмування представлений на малюнку 21.

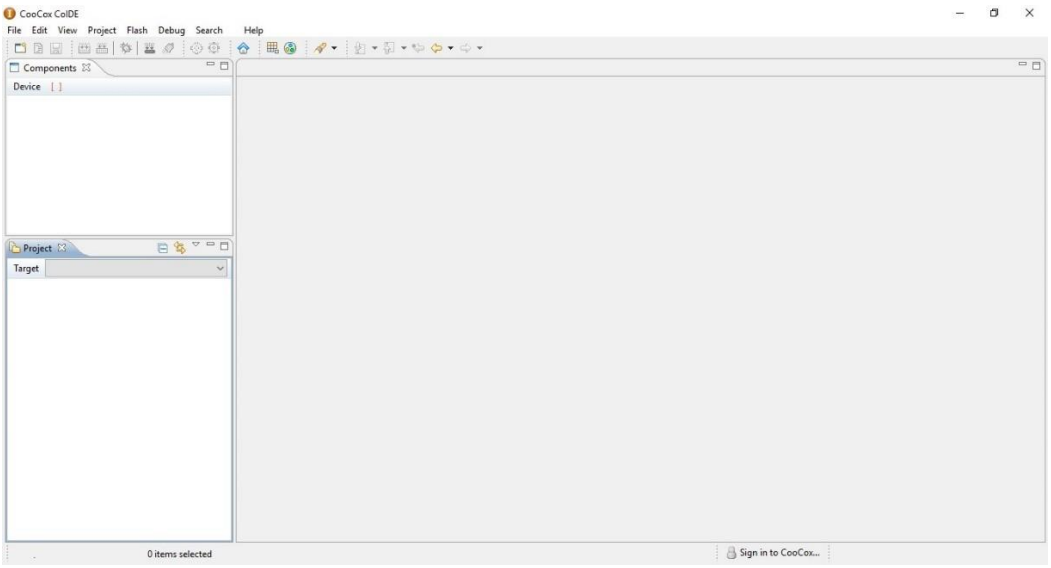


Рис. 21. Загальний вигляд CooCox CoIDE

Для роботи з CooCox CoIDE необхідно ознайомитися з панеллю інструментів (рисунок 22).

Рис. 22 - Панель інструментів

Основні інструменти, з якими працює користувач:

 - створити новий проект (New project);

 - створити новий файл (New file);

 - зберегти проект (Save);

 - зібрати проект, скомпілювати (Build);

 - почати налагодження пристрою (Start Debug);

 - завантаження коду до Flash-пам'яті (Download Code To Flash);

 - репозиторій, сховище, де знаходяться спільні бібліотеки (Repository).

Основний простір вікна займає текстовий редактор коду (рисунок 23), в ньому користувач набирає і редагує свої коди. У текстовому редакторі присутня вказівка на помилки, якщо код набрано невірно, з'являється напис про це. Також є дуже зручна система автодоповнення, яка, при написанні деякої послідовності символів, пропонує розробнику доповнити текст, щоб отримати необхідні функції у програмуванні.

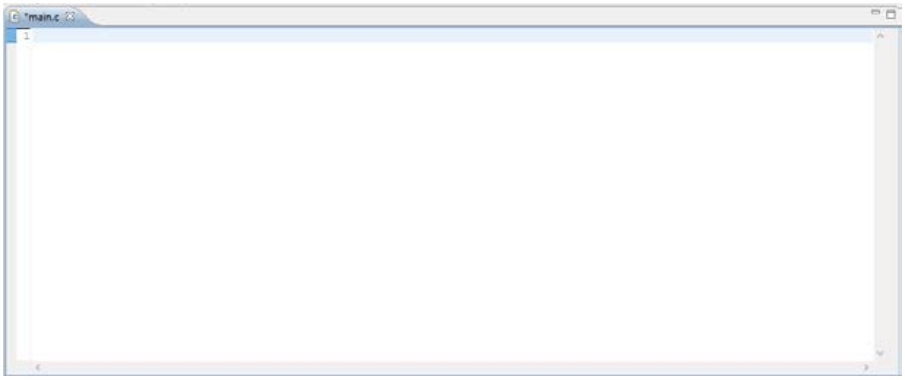


Рис. 23. Текстовий редактор коду CoIDE

Зліва від редактора коду знаходиться панель компонентів проекту і панель файлів, що входять до проекту (малюнок 24). У панелі компонентів відображаються всі підключені до проекту бібліотеки, а також до кожної з бібліотек можна подивитися приклади програм в інтернеті. В панелі файлів відображаються всі файли, які знаходяться в папці проекту і підключені до нього, можна підключати і створювати нові файли.

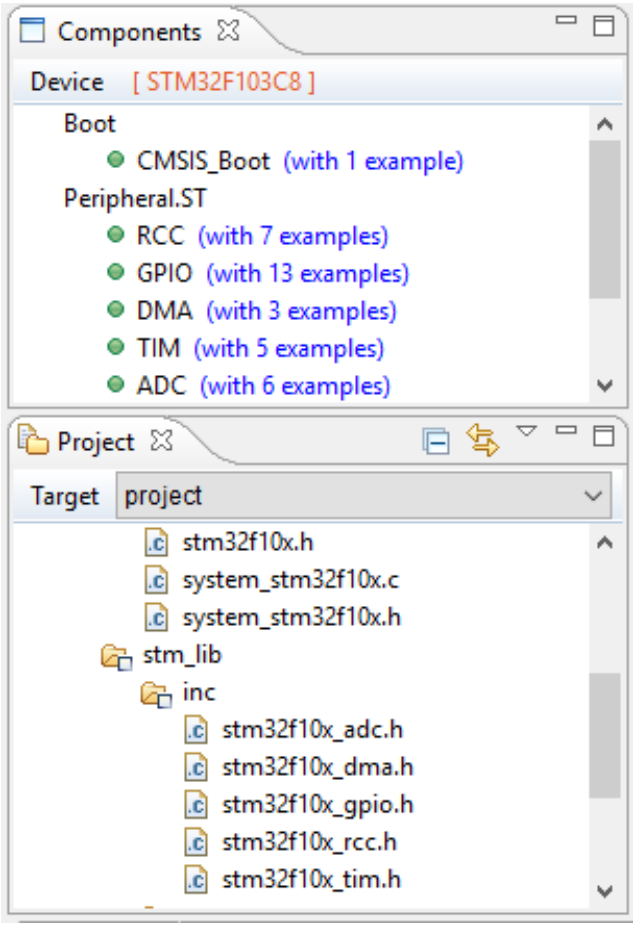


Рис. 24. Панель компонентів і панель файлів

Дуже важливою частиною вікна є консоль (рисунок 25). В консолі відображаються всі дії, які виконує користувач: компіляція, завантаження коду, налагодження і так далі.

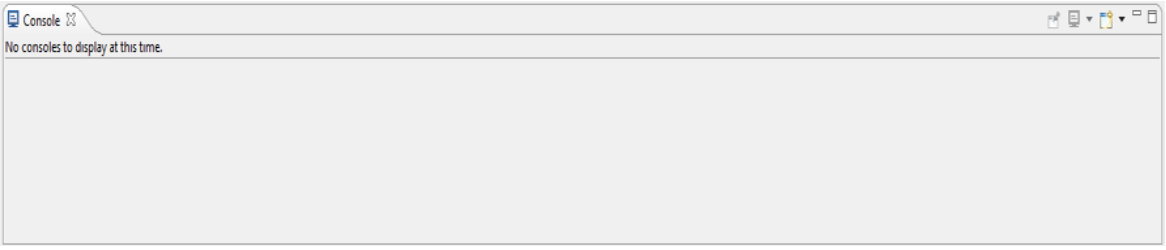


Рис. 25. Консоль

РОЗДІЛ ІІІ. РОЗРОБКА ЛАБОРАТОРНИХ РОБІТ З ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

3.1. Створення універсального автоматизованого робочого місця розробника програмного забезпечення мікроконтролерів

Для проведення лабораторних робіт був розроблений навчальний лабораторний стенд з використанням налагоджувальних плат на базі мікроконтролерів STM32. Зовнішній вигляд стенду представлений на рисунку 26.

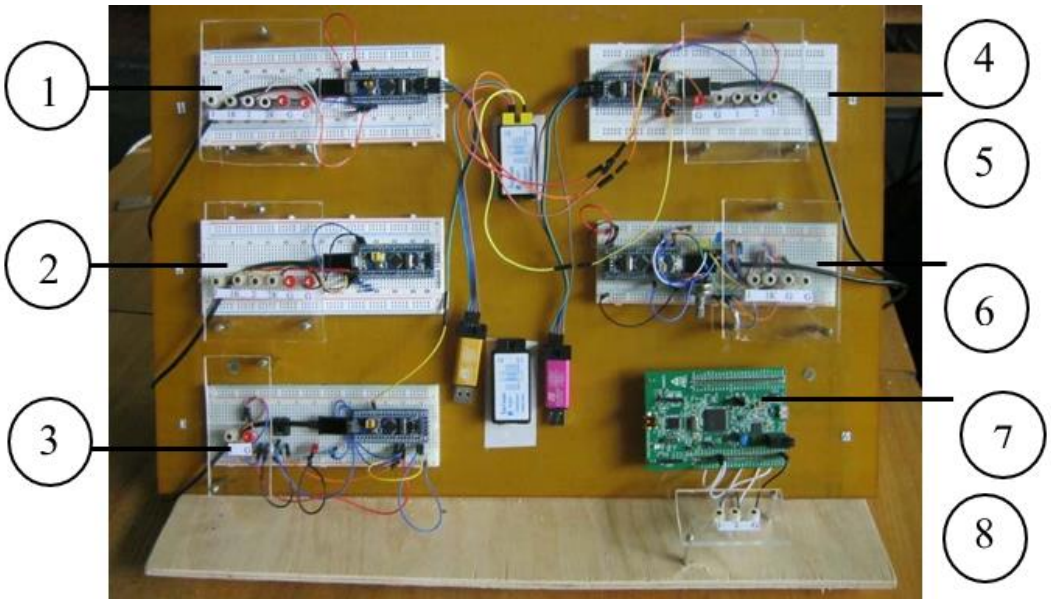


Рис. 26. Лабораторний стенд

Навчальний лабораторний стенд призначений для дослідження можливостей мікроконтролерів STM32, вивчення принципів програмування ШІМ-сигналів.

Компоненти лабораторного стенду:

1. Фрагмент лабораторного стенду з вивчення програмування синусоїдальних ШІМ-сигналів, зсунутих на 90°. Фрагмент виконано у вигляді макетної плати без пайки, зі встановленою на ній налагоджувальною платою на базі мікроконтролера STM32F103C8T6. Також до стенду на металевих стійках прикріплено оргскло, на ньому розташовані гнізда для підключення цифрового осцилографа. Використовується в лабораторній роботі № 1.

2. Фрагмент лабораторного стенду з вивчення програмування керуючих сигналів транзисторної стійки з генерацією мертвого часу (DeadTime). Так само, як і попередній, цей фрагмент виконаний у вигляді макетної плати без пайки, зі встановленою на ній налагоджувальною платою на базі мікроконтролера STM32F103C8T6. Також до стенду на металевих стійках прикріплено оргскло, на ньому розташовані гнізда для підключення цифрового осцилографа. Використовується в лабораторній роботі № 2.

3. Фрагмент лабораторного стенду з вивчення програмування регулювання шпаруватості ШІМ-сигналу за допомогою кнопки. Даний фрагмент виконаний у вигляді макетної плати без пайки, зі встановленою на ній налагоджувальною платою на базі мікроконтролера STM32F103C8T6, крім мікроконтролера, на макетній платі встановлена кнопка, при натисканні на яку змінюється вихідний сигнал, а також встановлений світлодіод, який наочно показує зміну шпаруватості імпульсу. Також до стенду на металевих стійках прикріплено оргскло, на ньому розташовані гнізда для підключення цифрового осцилографа. Використовується в лабораторній роботі № 3.

4. Фрагмент лабораторного стенду з вивчення програмування трьохфазних синусоїдальних ШІМ-сигналів, зсунутих на 120°. Виконаний у вигляді макетної плати без пайки, зі встановленою на ній

налагоджувальною платою на базі мікроконтролера STM32F103C8T6. Також до стенду на металевих стійках прикріплено оргскло, на ньому розташовані гнізда для підключення цифрового осцилографа. Використовується в лабораторній роботі № 4.

5. Фрагмент лабораторного стенду з вивчення програмування регулювання ШІМ-сигналу з допомогою потенціометра з генерацією мертвого часу (DeadTime). Фрагмент виконаний у вигляді макетної плати без пайки, зі встановленою на ній налагоджувальною платою на базі мікроконтролера STM32F103C8T6, крім мікроконтролера на макетній платі закріплений потенціометр, при обертанні якого змінюється шпаруватість вихідних імпульсів. Також до стенду на металевих стійках прикріплено оргскло, на ньому розташовані гнізда для підключення цифрового осцилографа. Використовується в лабораторній роботі № 5.

6. Фрагмент лабораторного стенду з вивчення програмування синусоїдального і пилкоподібного сигналів. У цьому фрагменті використовується налагоджувальна плата STM32F4-DISCOVERY (рисунок 27) на базі мікроконтролера STM32F407VGT6. Плата розміщена без використання макетної плати. Гнізда для підключення цифрового осцилографа розміщені на оргсклі. Використовується в лабораторній роботі № 6.

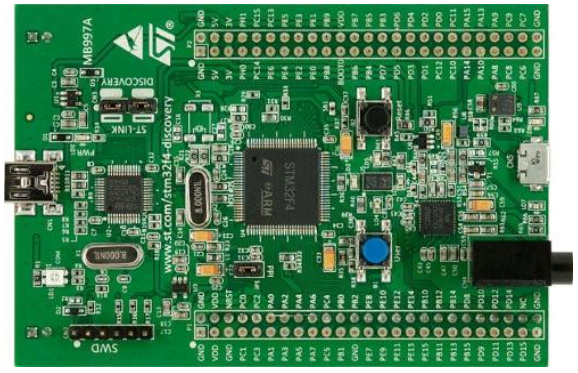


Рис. 27. Налагоджувальна плата STM32F4-DISCOVERY

7. Програматор ST-LINK / V2. Використовується для завантаження робочого коду до мікроконтролерів, може бути підключений до будь-якого з мікроконтролерів, розташованих на стенді. Також може бути використаний для налагодження програм.

8. Логічний аналізатор. Для того, щоб знімати вихідні сигнали з виводів плат, на лабораторному стенді розташовано два 8-канальні логічні аналізатори Saleae logic (рисунок 28). Логічний аналізатор - це електронний пристрій, здатний записувати і відображати послідовність цифрових сигналів; за допомогою спеціалізованого програмного забезпечення для персонального комп'ютера можливо проводити аналіз цифрових сигналів, що надходять з виводів мікроконтролера. До логічного аналізатора можливо підключати одночасно вісім виводів контролера.

Рис. 28. Логічний аналізатор Saleae logic

3.2. Лабораторна робота №1

Тема «Програмування двофазного генератора з синусоїдальними ШІМ-сигналами зі зсувом 90°»

Мета: програмування ШІМ-сигналів, зсунутих на 90°.

Устаткування:

– навчальний лабораторний стенд;

– персональний комп'ютер;

– двоканальний цифровий осцилограф.

Короткі теоретичні відомості:

Зрушення сигналів в даній лабораторній роботі проводиться за допомогою переривання, яке генерується системним таймером (SysTick). Системний таймер є частиною ядра Cortex-M3, це 24-бітний вираховувальний лічильник, з функціями автоперезавантаження і генерації переривання по завершенні рахунку. Даний таймер працює на тактовій частоті процесора.

Для ініціалізації системного таймера використовується функція SysTick_Config (), в аргументах (дужках) даної функції вказується кількість тактів, яку слід вважати таймером. Після ініціалізації системного таймера важливою частиною є опис функції переривання (обробника переривання). Всі функції переривання можна знайти в файлі startup_stm32f10x_md.c. На рисунку 29 представлений загальний вигляд функції переривання для системного таймера.

```
11 void SysTick_Handler(void)
12 {
13
14
15 }
```

Рис. 29. Функція переривання SysTick

Під час переривання процесор припиняє свою поточну активність, зберігаючи свій стан, і переходить до виконання обробника переривання. У обробнику переривання буде записаний код, в результаті якого буде отримана синусоїдальна напруга, зрушена на 90°.

Зміст і порядок виконання роботи:

1. На робочому столі комп'ютера знайти і подвійним клацанням лівої кнопки миші на значку CooCox CoIDE запустити середовище програмування.
2. Після запуску CooCox CoIDE в рядку меню натиснути: Project -> New Project.
3. У вікні в полі Project name, ввести ім'я свого майбутнього проекту англійською мовою, після цього натиснути Next>.
4. Вибрати поле з написом Chip, після цього натиснути Next>.
5. З'явиться вікно зі списками різних фірм мікроконтролерів (рисунок 30), необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F103x, після цього знайти мікроконтролер STM32F103C8, обрати його лівим клацанням миші і натиснути Finish.

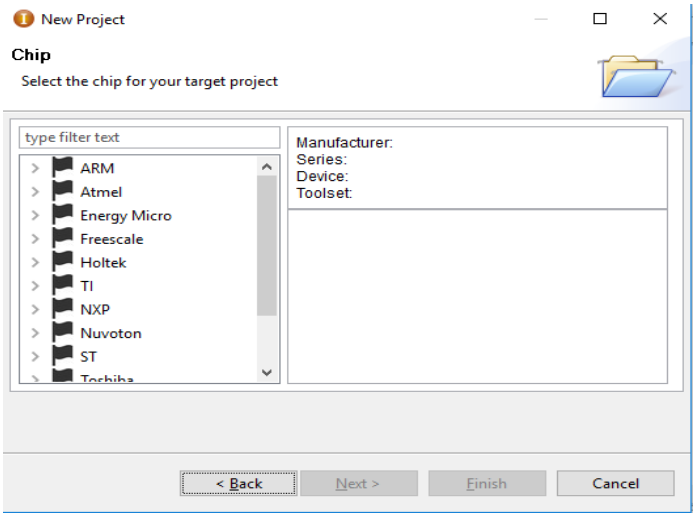


Рис. 30. Вибір мікроконтролера

6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору необхідних для проекту бібліотек, підключити наступні бібліотеки:

а) RCC - для управління тактовим генератором;

б) GPIO - для управління портами введення-виведення;

в) TIM - для управління таймерами.

7. Після вибору необхідних бібліотек, в панелі файлів вибрати файл main.c, весь код буде знаходитися тут.

8. За допомогою директиви #include <> підключити заголовки (рисунок 31).

```
#include <stm32f10x.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_tim.h>
```

Рис. 31. Підключення бібліотеки

9. Ввести всі структури, які будуть використовуватися в коді, а також ввести масив для побудови синусоїдальної ШІМ (рисунок 32). Масив має тип uint16_t, що означає, що даний масив не має негативних значень, а також числа, що входять в цей масив, можуть набувати значень в діапазоні від 0 до 65535.

```
//-------Ввод структур-------------
GPIO_InitTypeDef gpio;
GPIO_InitTypeDef gpio_1;
GPIO_InitTypeDef gpio_2;
GPIO_InitTypeDef gpio_3;
TIM_TimeBaseInitTypeDef TIM_Base_1;
TIM_OCInitTypeDef TIM_PWM;
TIM_BDTRInitTypeDef bdtr;
uint16_t sinPWM[125]={
        240,255,270,285,300,314,328,342,
        356,369,381,393,404,415,425,434,
        443,450,457,463,468,472,476,478,
        480,480,480,478,476,472,468,463,
        457,450,443,434,425,415,404,393,
        381,369,356,342,328,314,300,285,
        270,255,240,225,210,195,180,166,
        152,138,124,111,99,87,76,65,
        55,46,37,30,23,17,12,8,
        4,2,0,0,0,2,4,8,
        12,17,23,30,37,46,55,65,
        76,87,99,111,124,138,152,166,
        180,195,210,225,240,
        240,255,270,285,300,314,328,342,
        356,369,381,393,404,415,425,434,
        443,450,457,463,468,472,476,478
};
```

Рис. 32. Введення структур і масиву синуса

10. Для зручності сприйняття коду, програму було розділено на декілька підпрограм (функцій), спочатку необхідно ввести і заповнити функцію void initRcc (void) (рисунок 33) для включення тактування всіх використовуваних периферійних пристроїв. Ця функція має тип void.

```
//--------Функція включення тактування переферії------
void initRCC(void)
{
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPAEN
            | RCC_APB2ENR_TIM1EN | RCC_APB2ENR_AFIOEN;
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
}
```

Рис. 33. Функція включення тактування

11. Ввести і заповнити функцію ініціалізації всіх периферійних пристроїв void initAll (void), дана функція, як і попередня, також має тип void. Але так як вона має великий обсяг, необхідно розбити її на ділянки:

а) призначення портів введення-виведення (рисунок 34). Всі виводів назначені як альтернативні функції з двома станами;

```
//--------Призначення портів GPIO(Портів вводу-виводу)-----------
GPIO_StructInit(&gpio);
gpio.GPIO_Mode = GPIO_Mode_AF_PP;
gpio.GPIO_Pin = GPIO_Pin_8;
gpio.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &gpio);
//-----------------------------------
GPIO_StructInit(&gpio_1);
gpio_1.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_1.GPIO_Pin = GPIO_Pin_9;
gpio_1.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &gpio_1);
//-----------------------------------
GPIO_StructInit(&gpio_2);
gpio_2.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_2.GPIO_Pin = GPIO_Pin_13;
gpio_2.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &gpio_2);
//-----------------------------------
GPIO_StructInit(&gpio_3);
gpio_3.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_3.GPIO_Pin = GPIO_Pin_14;
gpio_3.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &gpio_3);
```

Рис. 34. Призначення GPIO в функції initAll ()

б) ініціалізація таймера TIM1 (рисунок 35):

```
79      //--------Призначення TIM1------------
80      TIM_TimeBaseStructInit(&TIM_Base_1);
81      TIM_Base_1.TIM_Prescaler = 0;
82      TIM_Base_1.TIM_CounterMode = TIM_CounterMode_Up;//таймер
83      TIM_Base_1.TIM_Period = 480;
84      TIM_Base_1.TIM_ClockDivision = TIM_CKD_DIV1;
85      TIM_TimeBaseInit(TIM1,&TIM_Base_1);
86      TIM_Cmd(TIM1, ENABLE);
```

Рис. 35. Ініціалізація таймера

в) ініціалізація ШІМ (рисунок 36)

```
87      TIM_OCStructInit(&TIM_PWM);
88      TIM_PWM.TIM_OCMode = TIM_OCMode_PWM1;
89      TIM_PWM.TIM_OutputState = TIM_OutputState_Enable; //Включення каналів ШІМ
90      TIM_PWM.TIM_OutputNState = TIM_OutputNState_Enable; //включення комплементарних каналів ШІМ
91      TIM_OC1Init(TIM1, &TIM_PWM);//перший ШІМ канал
92      TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
93      TIM_OC2Init(TIM1, &TIM_PWM);//другий ШІМ канал
94      TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

Рис. 36. Ініціалізація ШІМ

г) необхідно ініціалізувати системний таймер, для цього написати функцію SysTick_Config (), в аргументах у даній функції вказується кількість тактів, необхідно вказати 1200, в результаті повинно вийти SysTick_Config (1200).

12. Заповнити обробник переривання SysTick_Handler (рисунок 37), в якому проходитиме процес запису значень масиву в регістри порівняння таймера.

```
106 void SysTick_Handler(void) //функція переривання(системний таймер)
107 {
108     t1++;
109     if(t1>=100)
110     {t1=0;}
111     GPIO_ResetBits(GPIOA, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10); //скидання бітів
112     TIM1->CCR1 = (sinPWM[t1+25]);//запис данних із масива sinPWM і регістр порівняння CCR1 таймера TIM1
113     TIM1->CCR2 = (sinPWM[t1]); //запис данних із масива sinPWM в регістр порівняння CCR2
114
115 }
```

Рис. 37. Опис функції переривання

13. Після того, як всі функції були введені і заповнені, потрібно оголосити їх перед функцією int main (), як це робиться, представлено на рисунку 38.

```
33 void initRCC(void);
34 void initAll(void);
35 int main(void)
36 {
37
```

Рис. 38. Оголошення функцій

14. Записати дані функції між фігурними дужками в main () (рисунок 39), цикл while, в цьому проекті залишиться порожнім. Після цього код можна вважати завершеним.

```
35 int main(void)
36 {
37
38     initRCC();
39     initAll();
40
41     while(1)
42     {
43
44     }
45 }
```

Рис. 39. Функція main ()

15. Після написання коду програми його необхідно скомпілювати, для цього в панелі інструментів потрібно натиснути «Build», проект буде збиратися. У разі успішної компіляції, в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо ж в коді наявні помилки, тоді в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».

16. Після завершення компіляції, останнім етапом стане завантаження робочої програми до мікроконтролера (прошивка). Для цього потрібно через спеціальний кабель (подовжувач USB), підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в

панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження, в разі вдалого завантаження, в консолі з'являться написи: Erase: Done; Program: Done; Verify: Done. Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection», необхідно перевірити кабель, до якого він підключений.

17. Для перевірки працездатності програми необхідно підключити двоканальний цифровий осцилограф «GWINSTEK GDS-71062», до виводів на лабораторному стенді. Щупи осцилографа приєднуються до клем. Перший канал осцилографа необхідно підключити до клеми «G» і «1», попередньо налаштувавши масштаб 2В на осцилографі. Другий канал осцилографа необхідно підключити до клеми «G» і «2».

Опис клем представлено на рисунку 40.

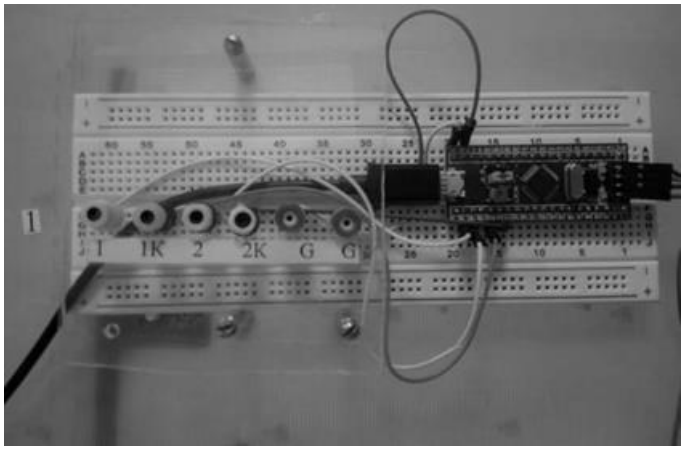


Рис. 40. Лабораторний стенд для дослідження зсуву двох сигналів на 90°

1 - висновок першого синусоїдального сигналу, 1К - висновок першого комплементарного сигналу, 2 - висновок другого синусоїдального сигналу, 2К - висновок другого комплементарного сигналу, G - земля.

Завдання: відповідно до призначеного викладачем варіантом, необхідно до кожного значення масиву синусів sinPWM додати відповідне значення з таблиці 6, скомпілювати отриманий проект, завантажити робочу

програму до мікроконтролера, отримати результат і продемонструвати його викладачеві.

Таблиця 6. Значення для таблиці синусів

варіант	1	2	3	4
значення	50	60	70	80

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.
3. Компонування елементів лабораторної установки, назва елементів, що входять до неї, у тому числі основних вузлів мікроконтролера.
4. Висновки про виконану роботу.
5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Що таке компіляція?
2. Дайте визначення терміну «переривання».
3. Поясніть, як в даному коді організовано зрушення сигналів на визначений електричний кут.

3.3. Лабораторна робота №2

Тема: «Програмування керуючих сигналів транзисторної стійки з генерацією «мертвого» часу (DeadTime)»

Мета: програмування ШІМ-сигналів з налаштуванням «мертвого часу».

Опис лабораторної роботи:

У процесі виконання даної лабораторної роботи студенти навчаться володіти технологією розробки програмного забезпечення для мікроконтролерів STM32 та познайомляться з методом генерації мертвого часу. Також наживо побачать результати виконаної роботи, що дозволить наочно переконатися в правильності виконання завдання.

Короткі теоретичні відомості:

«Мертвий» час (DeadTime) - затримка за часом позитивних фронтів керуючих сигналів для вимкнення аварійних ситуацій в стійках. Стійка (рисунок 41) - це основний елемент силової схеми, що складається з двох послідовно з'єднаних транзисторів і зворотних діодів, з'єднаних паралельно з ними.

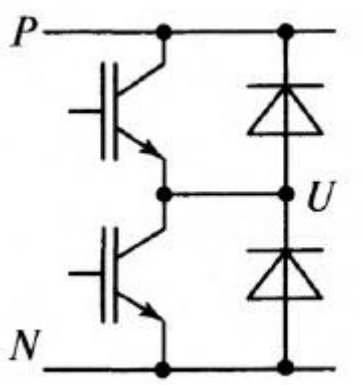


Рис. 41. Один з варіантів виконання стійки

Керування стійкою відбувається за допомогою ШІМ, зазвичай для стійки з двома транзисторами шпаруватість задається тільки для верхнього ключа, а нижній ключ працює в комплементарному режимі (рисунок 42),

тобто, коли верхній ключ увімкнений, нижній – вимкнений, і навпаки, коли нижній ключ увімкнений, верхній повинен бути вимкненим. Такий комплементарний спосіб управління застосовується в більшості перетворювачів. Мікроконтролер STM32F103C8T6 має в своєму наборі таймер TIM1, за допомогою якого можна призначити комплементарні виводи для генерації ШІМ-сигналу, тобто програмісту необхідно задати шпаруватість ШІМ тільки для верхнього ключа, а на відповідному висновку мікроконтролера апаратно сформується комплементарний сигнал для нижнього ключа.

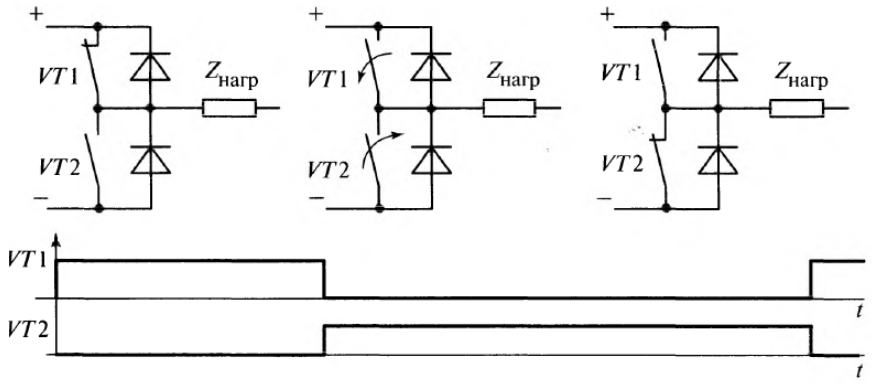


Рис. 42. Комплементарне управління транзисторами в стійці

Особливу увагу слід звернути на момент вимкнення верхнього транзистора і увімкнення нижнього. На практиці час спрацьовування транзисторів відмінний від нуля і не виключена ситуація, за якої один транзистор вже встиг увімкнутися, а інший ще не встиг вимкнутися. Це призводить до короткого замикання між позитивним і негативним контактами стійки. Струм, який виникає в такому аварійному режимі, називають «наскрізним». Для запобігання таких ситуацій використовують генерацію «мертвого» часу, тобто відбувається зсув фронтів сигналів (малюнок 43) і виникають паузи в управлінні, які гарантують безпечне вмикання та вимикання транзисторів в стійці.

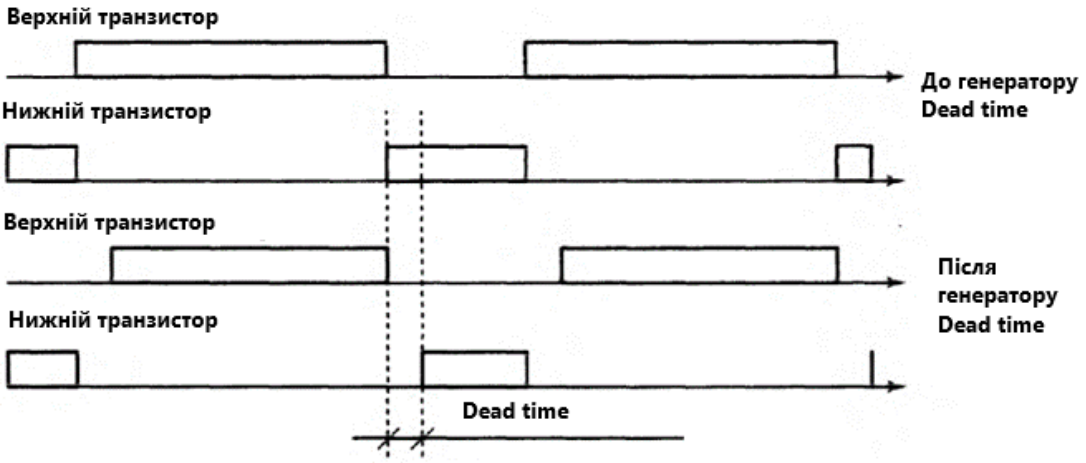


Рис. 43. Генерація «мертвого» часу

Зміст і порядок виконання роботи:

Представлено в Додатку А.

Завдання: відповідно до призначеного викладачем варіантом необхідно змінити значення поля TIM_DeadTime структури BDTR на значення з таблиці 7, скомпілювати отриманий проект, завантажити робочу програму до мікроконтролера і продемонструвати результат.

Таблиця 7. Значення для поля TIM_DeadTime

варіант	1	2	3	4
значення	20	30	40	50

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.
3. Компонування елементів лабораторної установки, назва елементів, які входять до неї, у тому числі основних вузлів мікроконтролера.
4. Висновки про виконану роботу.
5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Що таке мертвий час (DeadTime)?

2. Що таке наскрізний струм?

3. Поясніть, як відбувається управління стійкою за допомогою ШІМ.

4. Поясніть спосіб комплементарного управління стійкою.

5. Вкажіть, в якому полі структури BDTR вказується значення мертвого часу?

3.4. Лабораторна робота №3

Тема «Зміна шпаруватості ШІМ-сигналу за допомогою кнопки (Button)»

Мета: написати програму для зміни шпаруватості ШІМ-сигналу за допомогою кнопки.

Опис лабораторної роботи:

В процесі виконання даної лабораторної роботи студенти навчаться володіти технологією розробки програмного забезпечення для мікроконтролерів STM32. Познайомляться з особливостями ініціалізації такого периферійного пристрою як порт введення-виведення. Навчаться використовувати виводи мікроконтролера для зняття вхідних даних, в даному випадку з кнопки. Знайомство з портами введення-виведення - найбільш важлива частина в процесі вивчення мікроконтролерів, так як саме вони використовуються для взаємодії дії контролера із зовнішнім світом. Після виконаної роботи студенти будуть краще орієнтуватися в питаннях про розробки систем управління для «розумного» будинку, електроприводу і так далі.

Короткі теоретичні відомості:

Порти введення-виведення (GPIO) - це основний елемент будь-якого мікроконтролера, вони використовуються для «спілкування» мікроконтролера із зовнішнім світом. Пристрій портів введення-виведення представлено на малюнку 44.

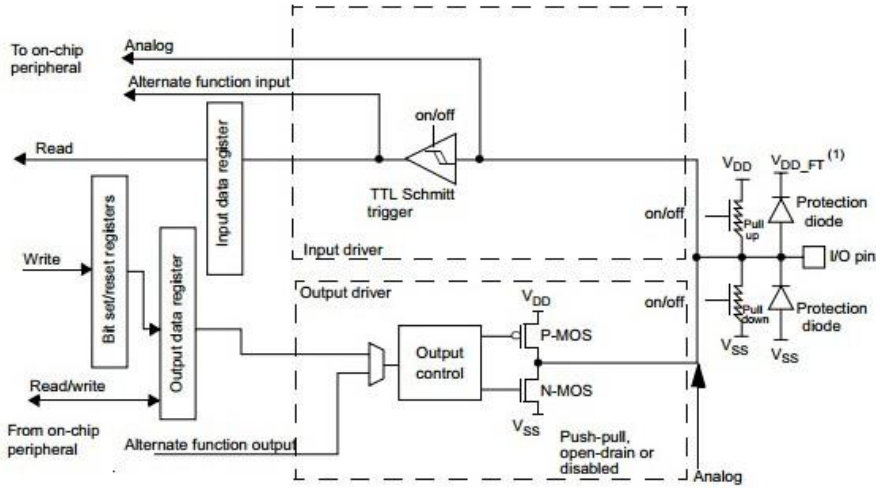


Рис. 44. Улаштування портів введення-виведення (GPIO)

Як видно з малюнка 44, виводи мікроконтролера можна конфігурувати як на вхід, так і на вихід:

1. Вхідний драйвер (input driver) - дозволяє налаштувати порт мікроконтролера як вхід (аналоговий вхід, альтернативна функція), з можливістю читання вхідних даних, і включає в себе:

а) тригер Шмітта (TTL Schmitt trigger);

б) стягувальний резистор (pull-down) – під’єднаний до виводу G (земля), дозволяє отримати на виведенні низьку напругу (логічний нуль);

в) підтягуючий резистор (pull-up) - підключений до живлення мікроконтролера, дозволяє отримати на виведенні високу напругу (логічну одиницю);

г) захисний діод (protection diode) - захищає мікроконтролер від перенапруги.

2. Вихідний драйвер (output driver) - дозволяє налаштувати порт мікроконтролера як вихід, з можливістю читання і запису вихідних даних, також можлива конфігурація порту у вигляді виходу альтернативної функції (наприклад, для генерації ШІМ-сигналів). Вихідний драйвер включає в себе:

а) управління виходом (output control);

б) транзистори (P - MOS і N - MOS), використовуються для визначення на виході мікроконтролера високої напруги (логічної одиниці), або низької напруги (логічного нуля);

в) захисний діод (protection diode) - захищає мікроконтролер від перенапруги.

У даній лабораторній роботі необхідно підключати кнопку. Кнопка - це пристрій, при натисканні на який відбувається замикання контактів. Для підключення кнопки до мікроконтролеру необхідно налаштувати порт мікроконтролера як вхід і включити читання вхідних даних.

Зміст і порядок виконання роботи:

Представлено в Додатку Б.

Завдання: відповідно до призначеного викладачем варіанту, необхідно змінити значення макросу TIM_PULSE_BUTTON_ON на значення з таблиці 8, скомпілювати проект, завантажити робочу програму в мікроконтролер, результат продемонструвати.

Таблиця 8 - Значення для макросу

варіант	1	2	3	4
значення	1600	1700	1800	2000

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.
3. Компонування елементів лабораторної установки, назва елементів, котрі входять до неї, у тому числі основних вузлів мікроконтролера.
4. Висновки про виконану роботу.
5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Що таке кнопка?

2. Напишіть фрагмент коду зчитування даних з виводу мікроконтролера. Для чого використовується підтягуючий до живлення резистор?

3. Напишіть фрагмент коду для ініціалізації виведення мікроконтролера як «входу».

3.5. Лабораторна робота №4

Тема «Програмування трьохфазного генератора з синусоїдальними ШІМ-сигналами зі зсувом 120°»

Мета: програмування ШІМ-сигналів, зсунутих на 120°.

Опис лабораторної роботи:

У процесі виконання даної лабораторної роботи студенти навчаться володіти технологією розробки програмного забезпечення для мікроконтролерів STM32. Познайомляться з одним із методів формування трьхфазних синусоїдальних ШІМ-сигналів, зсунутих на 120°, дізнаються про такі периферійні пристрої як DMA - прямий доступ до пам'яті, також дізнаються, як використовувати його, щоб отримувати вихідні сигнали заданої форми. Дана лабораторна робота дозволить студентам глибше зрозуміти принципи роботи частотних перетворювачів і сформованих в них ШІМ-сигналів.

Короткі теоретичні відомості:

В основі даної лабораторної роботи лежить принцип формування синусоїдальних ШІМ-сигналів, зсунутих на певний електричний кут. Таблиці для синусів можна розрахувати за такими формулами:

$$\begin{cases} \gamma_a = U_* sin(\omega_0 t); \\ \gamma_b = U_* sin\left(\omega_0 t + \frac{2\pi}{3}\right); \\ \gamma_b = U_* sin\left(\omega_0 t - \frac{2\pi}{3}\right), \end{cases}$$

,

де U * - відносне значення амплітуди напруги.

Для розрахунку даних таблиць можна використовувати будь-який розширений калькулятор, наприклад, можна скористатися засобами MS Excel.

Для передачі значень з масивів синусів в регістри порівняння таймера використовуються канали DMA. DMA - це прямий доступ до пам'яті,

обминаючи процесор, тобто можливо передавати величезні масиви даних, при цьому процесор буде «відпочивати». Це апаратний модуль, який розташовується на системній шині і має доступ до всієї пам'яті і до всіх периферійних пристроїв мікроконтролера.

Для того, щоб запустити DMA необхідно:

1. Включити модуль і канал DMA.
2. Вказати адресу джерела і адресу одержувача.
3. Задати деякі параметри передачі.

Після цього необхідно лише дати пристрою DMA команду «Старт» і передача даних почнеться. Незалежно від обсягу даних ця операція відбувається без участі процесора. У мікроконтролера STM32F103C8T6 в наявності є один 7-канальний модуль DMA.

Можливі три напрямки передачі даних по DMA:

1. Периферія-пам'ять, для прийому даних і складування їх у буфер.
2. Пам'ять-периферія, для передачі даних з буферу.
3. Пам'ять-пам'ять, копіювання блоку даних з однієї ділянки пам’яті в інший.

Зміст і порядок виконання роботи:

Представлено в Додатку В.

Завдання: відповідно до призначеного викладачем варіанту, необхідно змінити значення поля TIM_Prescaler структури *TIM_TimeBaseInitTypeDef* на значення з таблиці 9, скомпілювати проект, завантажити робочу програму в мікроконтролер, результат продемонструвати.

Таблиця 9. Значення для поля TIM_Prescaler

варіант	1	2	3	4
значення	5	10	15	20

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.
3. Компонування елементів лабораторної установки, назва елементів, що входять до неї, у тому числі основних вузлів мікроконтролера.
4. Висновки про виконану роботу.
5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Поясніть, як в даному коді організовано зрушення сигналів на визначений електричний кут.
2. Що таке DMA?
3. Назвіть напрямки передачі DMA.
4. За якими формулами будуються таблиці для ШІМ-сигналів?

3.6. Лабораторна робота №5

Тема: «Регулювання шпаруватості сигналу за допомогою аналогового потенціометра з налаштуванням «мертвого» часу (DeadTime)»

Мета: використовувати АЦП (аналогово-цифровий перетворювач) для зняття даних з потенціометра для регулювання шпаруватості сигналу.

Опис лабораторної роботи:

У процесі виконання даної лабораторної роботи студенти навчаться володіти технологією розробки програмного забезпечення для мікроконтролерів STM32 з використанням аналогових датчиків, в даному випадку, потенціометра. Студенти також познайомляться з основами тактування мікроконтролера і вибором множника частоти, а також переддільником, це дуже важлива частина, так як тактування є основою, від налаштування тактування залежить робота периферійних пристроїв. Також студенти побачать залежність мертвого часу від частоти тактування. Дана робота дає глибоке розуміння основних аспектів частотного регулювання електроприводу, який знайшов своє застосування в багатьох областях.

Короткі теоретичні відомості:

Налаштування тактування завжди починається з вибору кварцового резонатора. За замовчуванням контролер тактується від внутрішнього кварцового резонатора (HSI) на 8 МГц, ця частота не може бути змінена. Але мікроконтролер STM32F103C8T6 може працювати на частоті до 72 МГЦ, тому на налагоджувальній платі встановлений зовнішній кварцовий резонатор (HSE) на 8МГц, при ініціалізації якого частоту можна змінити, помноживши її на якийсь коефіцієнт. PLL - множник частоти, коефіцієнт, на який множиться частота зовнішнього кварцового резонатора. В даному випадку, максимально можливий коефіцієнт 9. Далі налаштовуються дільники частоти на шинах AHB, APB1, APB2.

Дані з аналогових датчиків, в даному випадку, з потенціометра, завжди знімаються за допомогою АЦП. АЦП – аналогово-цифровий перетворювач -

це пристрій, який перетворює вхідний аналоговий сигнал у вихідний сигнал в дискретній (цифровій) формі. Виконано на основі компаратора. Компаратор - пристрій порівняння -електронна схема, яка бере на свої входи два аналогових сигнали, порівнює їх і в результаті видається сигнал вищого рівня напруги.

У мікроконтролері STM32F103C8T6 в наявності є два 12-розрядних АЦП. Дані з АЦП можна зчитувати в буфер за допомогою DMA, що реалізовано в даній роботі. Також запуск АЦП можна організувати по тригеру, як тригер, наприклад, можна використовувати таймер, тобто АЦП буде запускатися, коли таймер дорахував до певного значення. Дане АЦП є 18-канальним, шістнадцять зовнішніх каналів і два внутрішніх.

АЦП є одним з найбільш важливих периферійних пристроїв мікроконтролера, так як природа навколо нас не дискретна, а безперервна, тому всілякі датчики зазвичай видають аналоговий сигнал і для перекладу його в цифровий вигляд використовується аналогово-цифровий перетворювач.

Зміст і порядок виконання роботи:

Представлено в Додатку Г.

Завдання: відповідно до призначеного викладачем варіанта необхідно змінити значення поля TIM_DeadTime структури BDTR на значення з таблиці 10, скомпілювати отриманий проект, завантажити робочу програму до мікроконтролера і продемонструвати результат.

Таблиця 10. Значення для поля TIM_DeadTime

варіант	1	2	3	4
значення	150	170	200	255

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.

3. Компонування елементів лабораторної установки, назва елементів, що входять до неї, у тому числі основних вузлів мікроконтролера.

4. Висновки про виконану роботу.

5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Що таке АЦП?

2. Для чого використовується PLL?

3. Поясніть, в чому відмінність внутрішнього і зовнішнього кварцового резонатора в мікроконтролері STM32?

4. Яке максимальне значення множника частоти може бути для мікроконтролера, використовуваного в лабораторній роботі?

5. Яким чином організовано зняття даних з потенціометра в даній лабораторної роботі?

3.7. Лабораторна робота №6

Тема: «Програмування синусоїдального і пилкоподібного сигналу за допомогою цифро-аналогового перетворювача (ЦАП)»

Мета: використовувати цифро-аналоговий перетворювач (ЦАП) для генерації синусоїдального і пилкоподібного сигналу.

Опис лабораторної роботи:

У процесі виконання даної лабораторної роботи студенти навчаться володіти технологією використання ЦАП для генерації аналогових сигналів заданої форми. Познайомляться з особливостями налаштування цифро-аналогового перетворювача, дізнаються про використання ЦАП спільно з тригером (в даному випадку з таймером), а також спільно з DMA.

Короткі теоретичні відомості:

ЦАП (DAC) – цифро-аналоговий перетворювач, пристрій для перетворення вхідного дискретного коду на аналоговий сигнал. ЦАП є інтерфейсом між дискретним цифровим світом і реальним аналоговим.

ЦАП в STM32F407VGT6 має наступні характеристики:

- напруга від 0 до 3.3В;
- апаратна генерація шуму і трикутних імпульсів;
- два незалежні канали;
- можливо перемикатися між 8- і 12-розрядним режимами з вирівнюванням бітів по лівому або по правому краю.

ЦАП використовує два висновки налагоджувальної плати PA4 і PA5, саме ці висновки застосовані в даній лабораторній роботі.

ЦАП в STM32 можна використовувати як генератор псевдовипадкових чисел, в такому випадку на цифровому осцилографі можна буде спостерігати шум. Також можна задавати амплітуду трикутних імпульсів.

Синусоїдальний сигнал апаратно отримати не можна, тому рекомендується використовувати ЦАП спільно з DMA, для відправки значень синуса до регістру ЦАП. Також синусоїдальний сигнал можна згенерувати,

використовуючи переривання, і описати в обробнику переривання запис значень масиву синуса в регістр ЦАП.

Зміст і порядок виконання роботи:

Представлено в Додатку Д.

Завдання: відповідно до призначеного викладачем варіанту, необхідно змінити значення поля DAC_LFSRUnmask_TriangleAmplitude в структурі ЦАП для генерації пилкоподібних сигналів на значення з таблиці 11, проект скомпілювати, завантажити робочу програму в мікроконтролер. Результат роботи продемонструвати.

Таблиця 11. Значення для генерації пилкоподібного сигналу

варіант	1	2
значення	DAC_TriangleAmplitude_15	DAC_TriangleAmplitude_31
варіант	3	4
значення	DAC_TriangleAmplitude_127	DAC_TriangleAmplitude_511

Зміст звіту:

1. Назва та мета роботи.
2. Код зі зміненими параметрами.
3. Компонування елементів лабораторної установки, назва елементів, що входять до неї, у тому числі основних вузлів мікроконтролера.
4. Висновки про виконану роботу.
5. Підготуватися до усного опитування по лабораторній роботі.

Питання для самоконтролю:

1. Що таке ЦАП?
2. Яку розрядність має ЦАП в STM32F407VGT6?
3. Що таке тригер?

4. Яким чином в даному коді реалізована генерація синусоїдального сигналу?

ВИСНОВКИ

Результатом даної бакалаврської роботи стало створення навчального лабораторного стенду з програмування мікроконтролерів STM32.

Програмування мікроконтролерів - досить специфічна тема, яка вимагає не тільки теоретичної підготовки, а також постійних практичних навичок. Неможливо вивчити програмування, спираючись лише на відомості, отримані з різних теоретичних джерел, без регулярних практичних занять навички програмування розвиватися не будуть.

Результат виконаної роботи можна розділити на три великі етапи: теоретичний, конструктивний і методичний.

На першому етапі було розглянуто базові відомості про мікроконтролери, проаналізовано такі з них як КР 580, PIC, AVR та STM, вказано на причини участі у даній розробці саме мікроконтролерів фірми STMicroelectronics (у порівнянні із AVR). Було проведено знайомство з архітектурою, параметрами, перевагами та недоліками мікроконтролерів STM, сказано про їхні основні відмінності від мікроконтролерів інших фірм. Також один розділ присвячено основам програмування, прошивки мікроконтролерів STM32 та розробці схеми пристроїв на мікроконтролерах. Було здійснено опис середовища програмування.

Результатом конструктивного етапу стало створення лабораторного стенду з програмування мікроконтролерів STM32. На ньому розташовуються шість мікроконтролерів даної фірми. Стенд був розроблений з метою проведення на ньому лабораторних робіт для вивчення програмування мікроконтролерів STM32.

На третьому етапі був розроблений комплекс лабораторних робіт. До комплексу лабораторних робіт входять шість робіт з програмування мікроконтролерів, які можна проводити на навчальному лабораторному стенді з застосуванням персонального комп'ютера. Для зняття сигналів

використовується цифровий двоканальний осцилограф. Лабораторні роботи дозволяють ознайомитися з мікропроцесорною технікою, вивчити основи програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brown G., Discovering The STM32 Microcontroller: work is covered by the Creative Commons Attibution-NonCommercial- ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) license, 2013.

2. Datasheet STM32F103x8, STM32F103xB, DocID13587 Rev 17, 2015.

3. Reay D., Digital signal processing using the ARM Cortex – M4: Published by John Wiley & Sons, Inc., Hoboken, New Jersey Published simultaneously in Canada, 2016.

4. . Reference manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM® -based 32-bit MCUs, RM0008, 2015.

5. Martin T., The Designer's Guide to the Cortex – M: The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB 225 Wyman Street, Waltham, MA 02451, USA, 2013.

6. Noviello C., Mastering STM32, a step-by-step guide to the most complete ARM Cortex – M platform, using a free and powerful development environment based on Eclipse and GCC, 2015 – 2016

7. Yiu J., The Definitive Guide to ARM Cortex – M0 and Cortex – M0+ Processors: The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB 225 Wyman Street, Waltham, MA 02451, USA, 2015.

ДОДАТКИ

Додаток А

Алгоритм набору коду в середовищі розробки CooCox CoIDE полягає в наступному:

1. Запускаємо середовище програмування CooCox CoIDE.

2. Після запуску CooCox CoIDE в рядку меню натиснути: Project New Project.

3. У вікні в полі «Project name» ввести ім'я свого проекту.

4. Далі потрібно вибрати поле з написом «Chip».

5. З'явиться вікно з випадаючими списками різних фірм мікроконтролерів (рис. 45). Необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F103x, після чого знайти мікроконтролер STM32F103C8, вибрати його лівим клацанням миші і натиснути Finish (рис. 46).

6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору необхідних для проекту бібліотек (рис. 47).

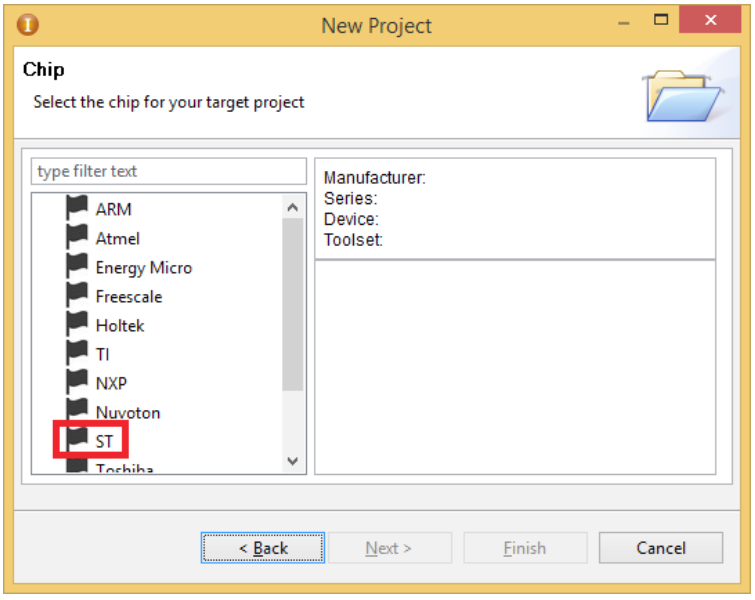


Рис. 45. Вибір фірми мікроконтролера

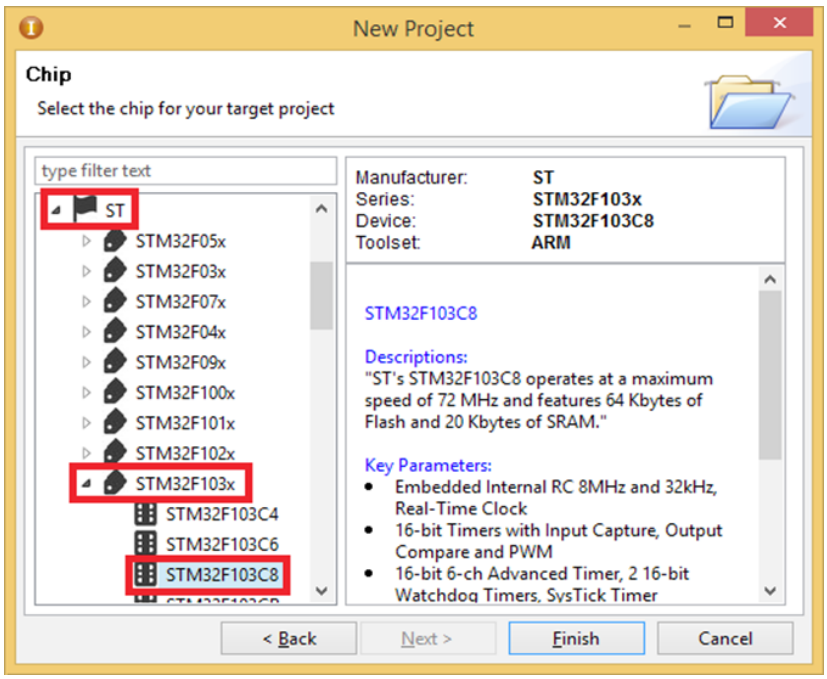


Рис. 46. Вибір мікроконтролера

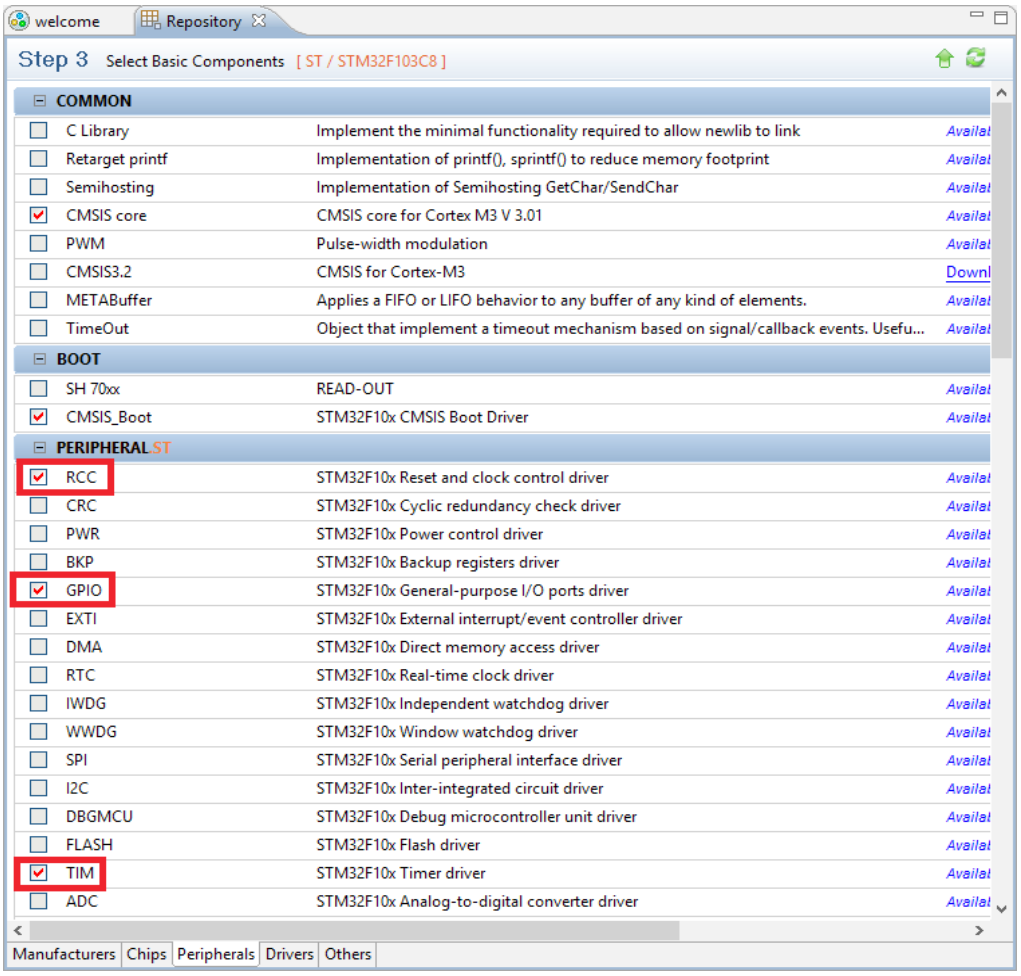


Рис. 47. Вибір бібліотек в репозиторії

Необхідно підключити наступні бібліотеки:

- RCC - для управління тактовим генератором;
- GPIO - для управління портами введення-виведення;
- TIM - для управління таймерами.

7. Після вибору необхідних бібліотек в панелі файлів потрібно вибрати файл «main. c », весь код буде знаходиться тут.

8. За допомогою директиви «#include <>« необхідно підключити заголовки (рис. 48).

```
1 #include <stm32f10x.h>
2 #include <stm32f10x_gpio.h>
3 #include <stm32f10x_rcc.h>
4 #include <stm32f10x_tim.h>
```

Рис. 48. Підключення заголовкових файлів

9. Далі потрібно ввести всі структури, які будуть використовуватися в коді, а також ввести масив для побудови синусоїдальної ШІМ (рис. 49). Масив має тип «uint16_t», що означає, що даний масив не має негативних значень, а також числа, що входять в цей масив, можуть набувати значень в діапазоні від 0 до 65535.

```
9  GPIO_InitTypeDef gpio;
10 GPIO_InitTypeDef gpio_1;
11 GPIO_InitTypeDef gpio_2;
12 GPIO_InitTypeDef gpio_3;
13 TIM_TimeBaseInitTypeDef TIM_Base_1;
14 TIM_OCInitTypeDef TIM_PWM;
15 TIM_BDTRInitTypeDef bdtr;
16 uint16_t sinPWM[125]={
17         240,255,270,285,300,314,328,342,
18         356,369,381,393,404,415,425,434,
19         443,450,457,463,468,472,476,478,
20         480,480,480,478,476,472,468,463,
21         457,450,443,434,425,415,404,393,
22         381,369,356,342,328,314,300,285,
23         270,255,240,225,210,195,180,166,
24         152,138,124,111,99,87,76,65,
25         55,46,37,30,23,17,12,8,
26         4,2,0,0,0,2,4,8,
27         12,17,23,30,37,46,55,65,
28         76,87,99,111,124,138,152,166,
29         180,195,210,225,240,
30         240,255,270,285,300,314,328,342,
31         356,369,381,393,404,415,425,434,
32         443,450,457,463,468,472,476,478
33 };
```

Рис. 49. Введення структур і масива синуса

10. Для зручності сприйняття коду програма була розділена на кілька підпрограм (функцій). Спочатку необхідно ввести і заповнити функцію «void initRcc (void)» (рис. 50) для увімкнення тактування всіх використовуваних периферійних пристроїв. Ця функція має тип «void».

```
46 void initRCC(void)
47 {
48     RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPAEN
49                   | RCC_APB2ENR_TIM1EN | RCC_APB2ENR_AFIOEN;
50     RCC->AHBENR |= RCC_AHBENR_DMA1EN;
51 }
```

Рис. 50. Функція увімкнення тактування

11. Наступним кроком буде введення і заповнення функції ініціалізації всіх периферійних пристроїв «void initAll (void)». Ця функція, як і попередня, також має тип «void». Але так як вона має великий обсяг, необхідно розбити її на ділянки:

- Призначення портів введення-виведення (рис. 51). Всі висновки призначені як альтернативні функції з двома станами;

```
56 GPIO_StructInit(&gpio);
57 gpio.GPIO_Mode = GPIO_Mode_AF_PP;
58 gpio.GPIO_Pin = GPIO_Pin_8;
59 gpio.GPIO_Speed = GPIO_Speed_50MHz;
60 GPIO_Init(GPIOA, &gpio);
61 //-----------------------------------
62 GPIO_StructInit(&gpio_1);
63 gpio_1.GPIO_Mode = GPIO_Mode_AF_PP;
64 gpio_1.GPIO_Pin = GPIO_Pin_9;
65 gpio_1.GPIO_Speed = GPIO_Speed_50MHz;
66 GPIO_Init(GPIOA, &gpio_1);
67 //-----------------------------------
68 GPIO_StructInit(&gpio_2);
69 gpio_2.GPIO_Mode = GPIO_Mode_AF_PP;
70 gpio_2.GPIO_Pin = GPIO_Pin_13;
71 gpio_2.GPIO_Speed = GPIO_Speed_50MHz;
72 GPIO_Init(GPIOB, &gpio_2);
73 //---------------------------------
74 GPIO_StructInit(&gpio_3);
75 gpio_3.GPIO_Mode = GPIO_Mode_AF_PP;
76 gpio_3.GPIO_Pin = GPIO_Pin_14;
77 gpio_3.GPIO_Speed = GPIO_Speed_50MHz;
78 GPIO_Init(GPIOB, &gpio_3);
```

Рис. 51. Призначення GPIO у функції initAll ()

- Ініціалізація таймера TIM1 (рис. 52);

```
80    TIM_TimeBaseStructInit(&TIM_Base_1);
81
82    TIM_Base_1.TIM_Prescaler = 0;
83
84    TIM_Base_1.TIM_CounterMode = TIM_CounterMode_Up;
85    TIM_Base_1.TIM_Period = 480;
86    TIM_Base_1.TIM_ClockDivision = TIM_CKD_DIV1;
87    TIM_TimeBaseInit(TIM1,&TIM_Base_1);
88    TIM_Cmd(TIM1, ENABLE);
```

Рис. 52. Ініціалізація таймера

- Для генерації «мертвого часу» необхідно заповнити всього два поля, як показано на рис. 53;

```
98    TIM_BDTRStructInit(&bdtr);
99    bdtr.TIM AutomaticOutput = TIM AutomaticOutput Enable;
100
101   bdtr.TIM_DeadTime =10;
102   TIM_BDTRConfig(TIM1, &bdtr);
103   TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

Рис. 53. Налаштування «мертвого часу»

- Ініціалізація ШІМ (рис. 54);

```
90    TIM_OCStructInit(&TIM_PWM);
91    TIM_PWM.TIM_OCMode = TIM_OCMode_PWM1;
92
93    TIM_PWM.TIM_OutputState = TIM_OutputState_Enable;
94
95    TIM_PWM.TIM_OutputNState = TIM_OutputNState_Enable;
96
97    TIM_OC1Init(TIM1, &TIM_PWM);
98    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
99
100   TIM_OC2Init(TIM1, &TIM_PWM);
101   TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

Рис. 54. Ініціалізація ШІМ

- Необхідно ініціювати системний таймер. Для цього потрібно написати функцію SysTick_Config (), в аргументах якої вказується частота тактування. Необхідно вказати частоту 1200, в результаті повинно вийти SysTick_Config (1200).

12. Після введення і заповнення функції тактування необхідно заповнити функцію переривання SysTick_Handler (рис. 55), в якій буде проходити процес запису значень масиву в регістри порівняння таймера.

```
107 void SysTick_Handler(void)
108 {
109     t1++;
110     if(t1>=100)
111     {t1=0;}
112
113     GPIO_ResetBits(GPIOA, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10);
114
115     TIM1->CCR1 = (sinPWM[t1+25]);
116
117     TIM1->CCR2 = (sinPWM[t1]);
118 }
```

Рис. 55. Опис функції переривання

13. Після тог,о як всі функції були введені і заповнені, потрібно оголосити їх перед функцією «int main ()» (рис. 56);

```
33 void initRCC(void);
34 void initAll(void);
35 int main(void)
36 {
37
```

Рис. 56. Оголошення функцій

14. Наступним кроком необхідно записати дані функції між фігурними дужками в «main ()» (рис. 57). Цикл «while» в цьому проекті залишиться порожнім. Після цього код можна вважати завершеним.

```
35 int main(void)
36 {
37
38     initRCC();
39     initAll();
40
41     while(1)
42     {
43
44     }
45 }
```

Рис. 57. Функція «main ()»

15. Після написання коду програми, його необхідно скомпілювати. Для цього в панелі інструментів потрібно натиснути «Build». У разі успішної компіляції в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо в коді присутні помилки, то в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».

16. Після завершення компіляції останнім етапом стане завантаження робочої програми до мікроконтролера. Для цього потрібно через спеціальний кабель (подовжувач USB) підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження. У разі вдалого завантаження в консолі з'являться написи: «Erase: Done»; «Program: Done»; «Verify: Done». Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection». Необхідно перевірити кабель, до якого він підключений.

На рис. 58 представлений загальний вигляд лабораторного стенду з мікроконтролерами STM32.

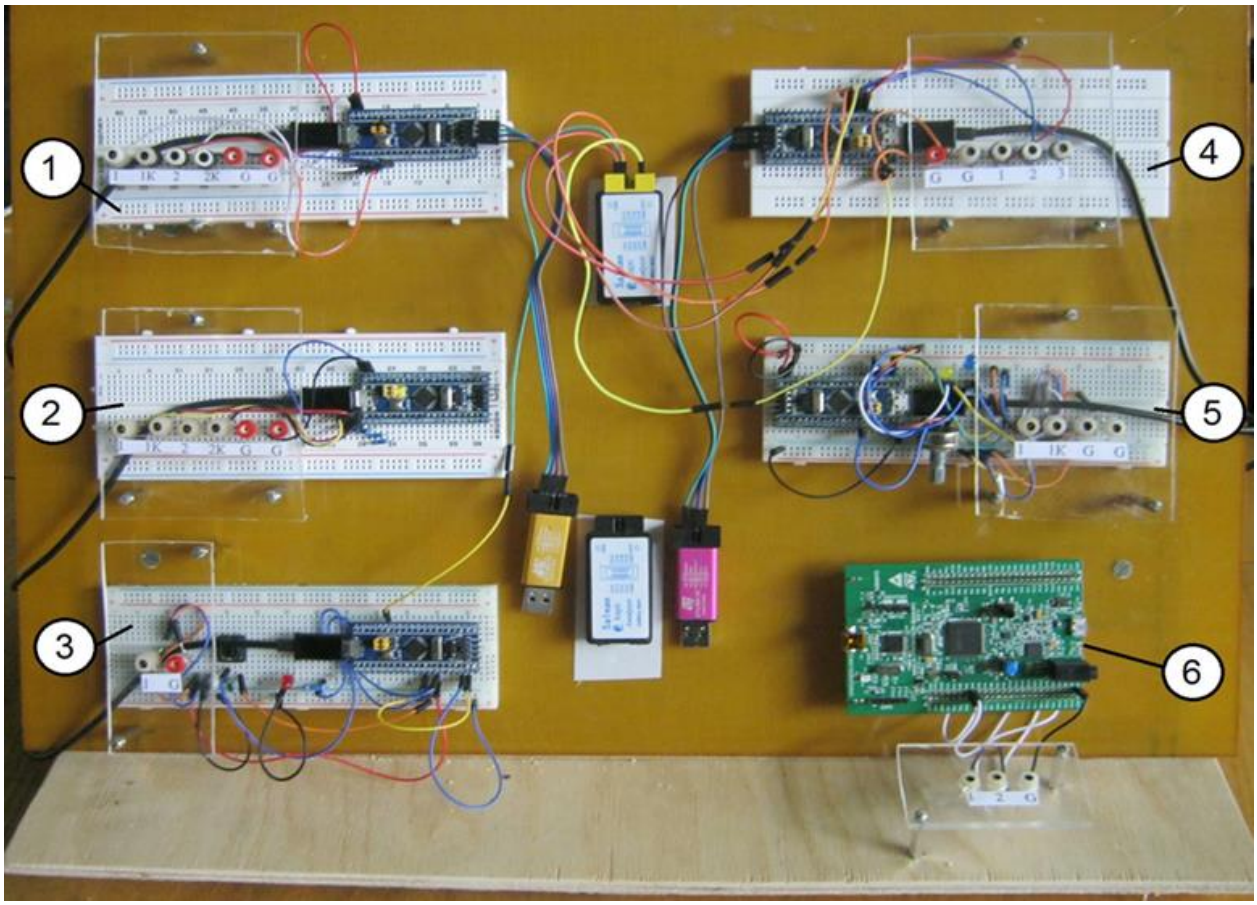


Рис. 58. Загальний вигляд лабораторного стенду для програмування мікроконтролерів STM32

- 1 - програмування двухфазного генератора з синусоїдальними напругами із зсувом 90 °;
- 2 - програмування прямого і комплементарного ШІМ-сигналів з налаштуванням «мертвого часу»;
- 3 - програмування зміни шпаруватості імпульсів ШІМ на мікроконтролері STM32 за допомогою кнопки;

- 4 - програмування трьохфазного генератора з синусоїдальними напругами із зсувом 120 °;
- 5 - регулювання шпаруватості сигналу за допомогою аналогового потенціометра з налаштуванням «мертвого часу».

У кожній монтажній платі є клеми для підключення до відповідних каналів цифрового осцилографа. У даній лабораторній роботі приведено зміст лабораторної роботи на монтажній платі з мікроконтролером під номером 2 (рис.58).

Осцилограма прямого і комплементарного ШІМ-сигналів з налаштуванням «мертвого часу» приведена на рис. 59.

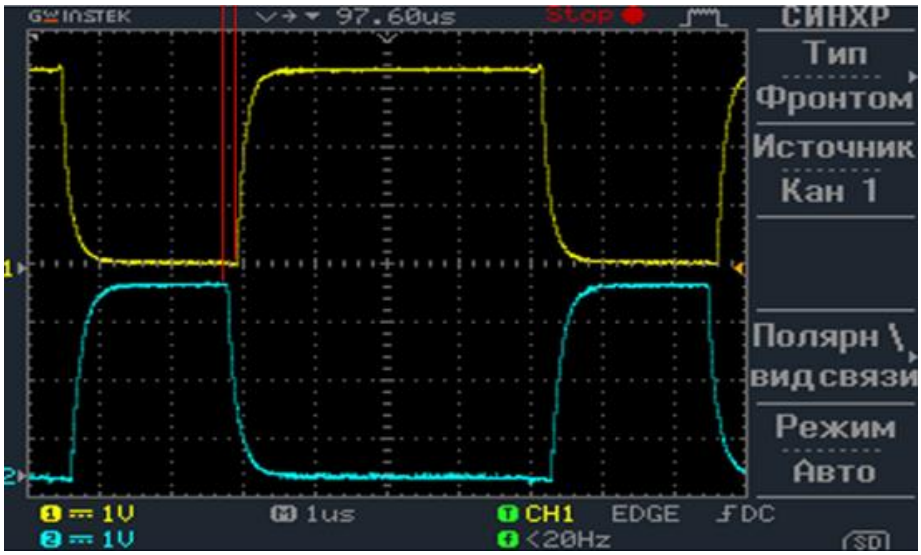


Рис. 59. Осцилограма прямого і комплементарного ШІМ-сигналів з налаштуванням «мертвого часу»

Додаток Б

Алгоритм набору коду в програмі CoIDE полягає в наступному:

1. Запускаємо середовище програмування CooCox CoIDE.

2. Після запуску CooCox CoIDE в рядку меню натиснути: Project → New Project.

3. У вікні в полі «Project name» ввести ім'я свого проекту.

4. Далі потрібно вибрати поле з написом «Chip».

5. З'явиться вікно з випадаючими списками різних фірм мікроконтролерів (рис. 60). Необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F103x, після чого знайти мікроконтролер STM32F103C8, вибрати його лівим клацанням миші і натиснути Finish (рис. 61).

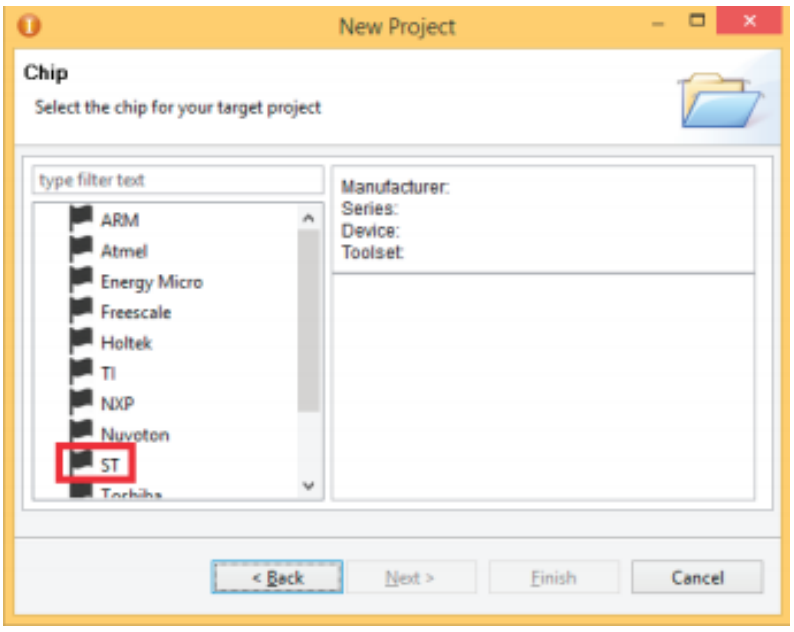


Рис. 60. Вибір фірми мікроконтролера

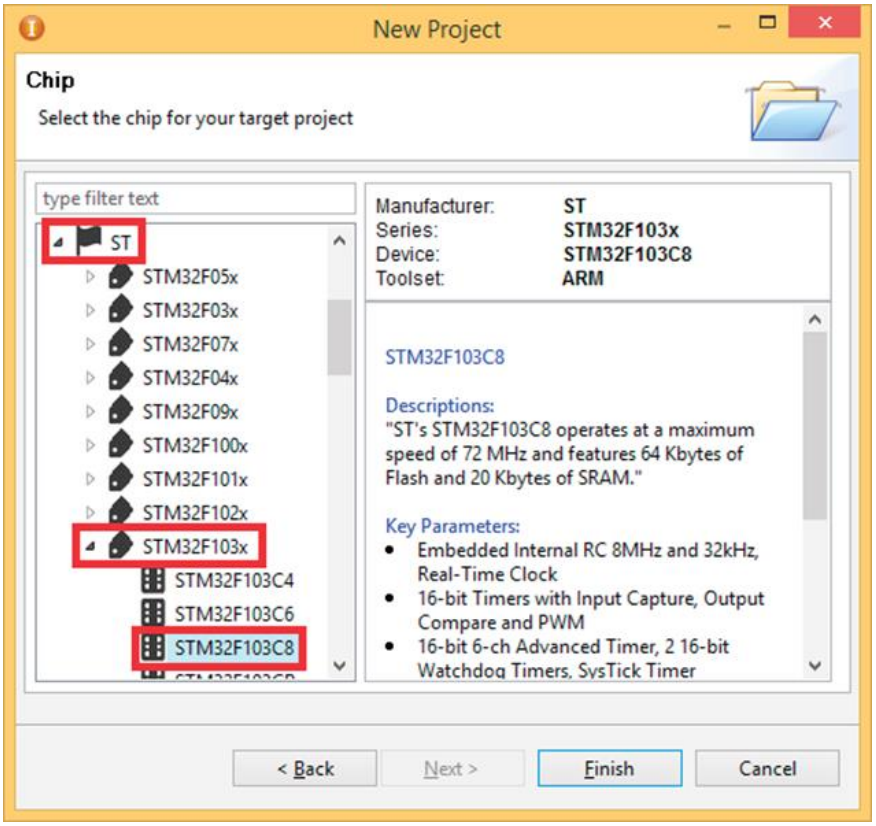


Рис. 61. Вибір мікроконтролера

6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору необхідних для проекту бібліотек (рис. 62). Необхідно підключити наступні бібліотеки:

1) RCC - для управління тактовим генератором;

2) GPIO - для управління портами введення-виведення;

3) TIM - для управління таймерами.

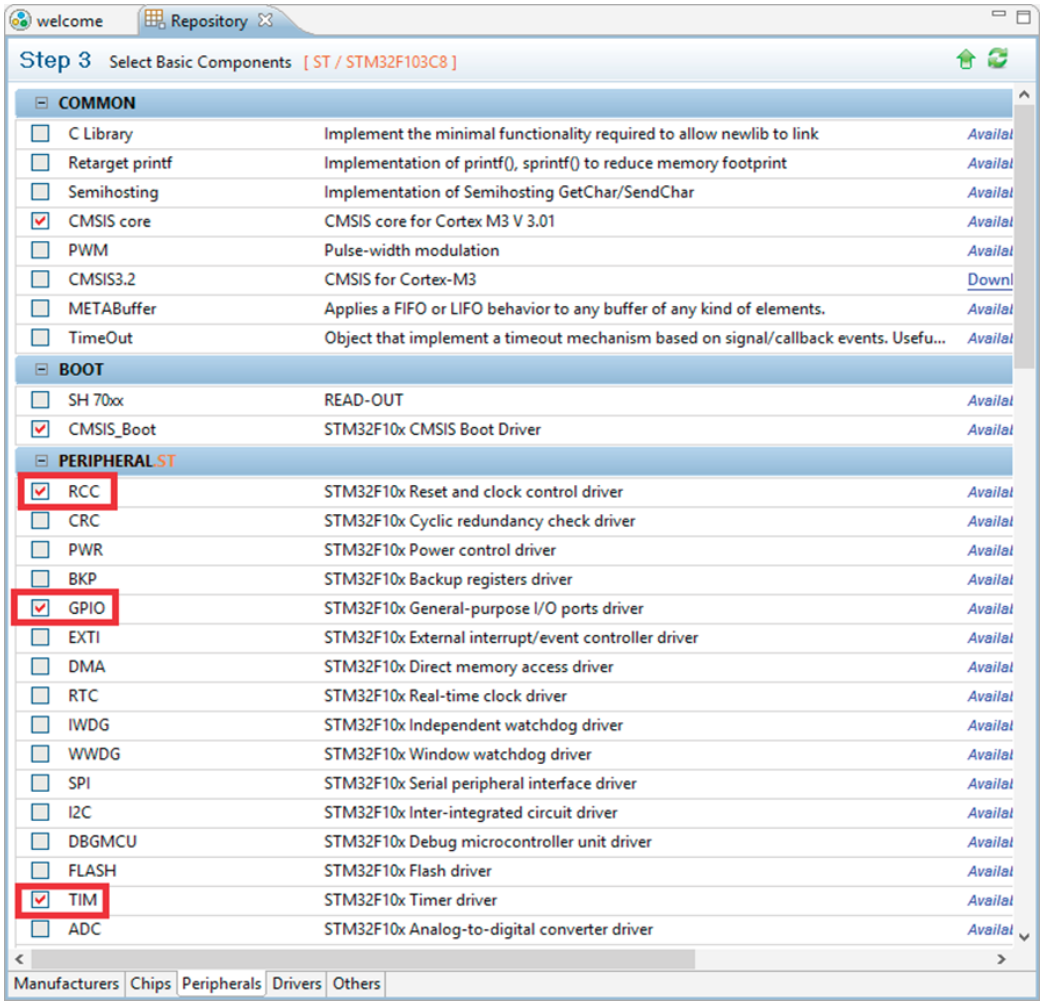


Рис. 62. Вибір бібліотек в репозиторії

7. Після вибору необхідних бібліотек в панелі файлів потрібно вибрати файл «main.c», весь код буде знаходитися тут.
8. За допомогою директиви «#include <>» необхідно підключити заголовки (рис. 63).

```
1 #include <stm32f10x.h>
2 #include <stm32f10x_gpio.h>
3 #include <stm32f10x_rcc.h>
4 #include <stm32f10x_tim.h>
```

Рис. 63. Підключення заголовкових файлів

9. Після підключення заголовкових файлів необхідно за допомогою директиви «#define» ввести значення для положень кнопки «ON» - натиснута і «OFF» - не натиснута (рис. 64).

```
6 #define TIM_PULSE_BUTTON_ON 1500
7 #define TIM_PULSE_BUTTON_OFF 750
```

Рис. 64. Введення значень для положень кнопки

10. Необхідно увімкнути тактування периферійних пристроїв. Для цього потрібно створити і заповнити функцію «InitRCC ()» (рис. 65).

```
41 void InitRCC(void)
42 {
43     RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;
44     RCC->APB2ENR |= RCC_APB2ENR_AFIOEN | RCC_APB2ENR_IOPBEN|RCC_APB2ENR_IOPCEN;
45 }
```

Рис. 65. Функція увімкнення тактування периферійних пристроїв

11. Наступним кроком буде створення і заповнення функції ініціалізації периферійних пристроїв «InitAll ()»:

1) Необхідно ввести структури для ініціалізації портів введення-виведення (GPIO) і таймера (рис. 66);

```
12 TIM_TimeBaseInitTypeDef timer;
13 GPIO_InitTypeDef led;
14 TIM_OCInitTypeDef timerPWM;
15 GPIO_InitTypeDef button;
```

Рис. 66. Введення структур

2) Провести ініціалізацію і налаштування портів введення-виведення (рис. 67);

```
17 GPIO_StructInit(&led);
18 led.GPIO_Pin = GPIO_Pin_6;
19 led.GPIO_Speed = GPIO_Speed_50MHz;
20 led.GPIO_Mode = GPIO_Mode_AF_PP;
21 GPIO_Init(GPIOB, &led);
22 GPIO_StructInit(&button);
23 button.GPIO_Pin = GPIO_Pin_4;
24 button.GPIO_Speed = GPIO_Speed_2MHz;
25 button.GPIO_Mode = GPIO_Mode_IN_FLOATING;
26 GPIO_Init(GPIOB,&button);
```

Рис. 67. Ініціалізація GPIO

3) Ініціалізація і налаштування таймера TIM4 (рис. 68);

```
28 TIM_TimeBaseStructInit(&timer);
29 timer.TIM_CounterMode= TIM_CounterMode_Up;
30 timer.TIM_Prescaler = 0;
31 timer.TIM_Period = 2000;
32 TIM_TimeBaseInit(TIM4, &timer);
```

Рис. 68. Ініціалізація таймера (TIM4)

4) Ініціалізація і налаштування ШІМ (рис. 69);

```
34 TIM_OCStructInit(&timerPWM);
35 timerPWM.TIM_Pulse=600;
36 timerPWM.TIM_OCMode = TIM_OCMode_PWM1;
37 timerPWM.TIM_OutputState = TIM_OutputState_Enable;
38 TIM_Cmd(TIM4, ENABLE);
39 TIM_OC1Init(TIM4, &timerPWM);
```

Рис. 69. Налаштування ШІМ

12. Після того, як функції «InitRCC ()» і «InitAll ()» були введені, необхідно оголосити їх перед функцією «Main ()» (рис. 70);

```
46 void InitAll(void);
47 void InitRCC(void);
48 int main(void)
49 {
50
```

Рис. 70. Оголошення функцій

13. Наступним кроком необхідно записати функціонально «InitRCC ()» і «InitAll ()» між фігурними дужками в функцію «main ()», а також в циклі «while (1)» за допомогою структури «if ... else» задати параметри для кнопки і регістрів порівняння таймера TIM4 (рис. 71). Після виконаних дій код можна вважати завершеним.

```
48 int main(void)
49 {
50
51 InitRCC();
52 InitAll();
53 while(1)
54     {
55
56     if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_4)==0x01)
57     {
58
59         TIM4->CCR1=TIM_PULSE_BUTTON_ON;
60     }
61     else
62     {
63
64         TIM4->CCR1 = TIM_PULSE_BUTTON_OFF;
65     }
66 }
```

Рис. 71. Заповнення функції «main()»

14. Після написання коду програми, його необхідно скомпілювати. Для цього в панелі інструментів потрібно натиснути «Build». У разі успішної компіляції в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо ж в коді присутні помилки, то в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».

15. Після завершення компіляції останнім етапом стане завантаження робочої програми в мікроконтролер. Для цього потрібно через спеціальний кабель (подовжувач USB) підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження. У разі

вдалого завантаження в консолі з'являться написи: «Erase: Done»; «Program: Done»; «Verify: Done». Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection». Необхідно перевірити кабель, до якого він підключений.

Результати зміни шпаруватості імпульсів ШІМ за допомогою кнопки подані на рис. 72:

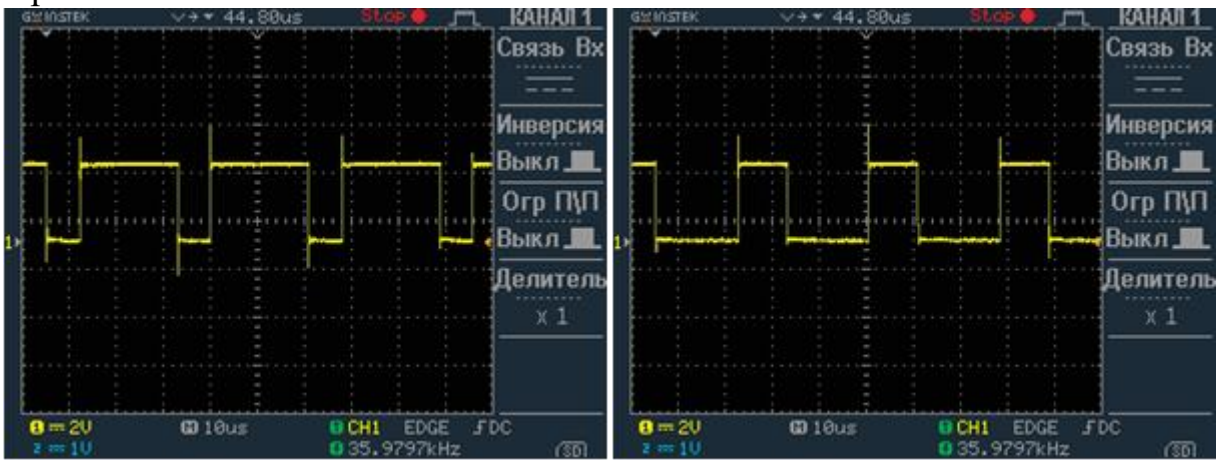


Рис. 72. Осцилограми зміни шпаруватості імпульсів ШІМ за допомогою кнопки

Додаток В

1. Запускаємо середовище програмування CooCox CoIDE.
2. Після запуску CooCox CoIDE в рядку меню натиснути: Project →New Project.
3. У вікні в полі «Project name» ввести ім'я свого проекту.
4. Далі потрібно вибрати поле з написом «Chip».
5. З'явиться вікно з випадаючими списками різних фірм мікроконтролерів (рис. 73). необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F103x, після чого знайти мікроконтролер STM32F103C8, вибрати його лівим клацанням миші і натиснути Finish (рис. 74).
6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору необхідних для проекту бібліотек (рис. 75). Необхідно підключити наступні бібліотеки:
- RCC-для управління тактовим генератором;
- GPIO-для управління портами введення-виведення;
- TIM-для управління таймерами;
- DMA-для керування прямим доступом до пам'яті (DMA).
7. Після вибору бібліотек необхідно на панелі інструментів вибрати «New file» і створити файли «timer» і «sinDMA» з розширенням «.s», а також аналогічні файли з розширенням «.h» (рис. 76).
8. Потім необхідно в панелі файлів відкрити файл «timer.h», два рази клацнувши по ньому лівою кнопкою миші, і за допомогою директиви «#include <>» записати в ньому всі заголовки, необхідні для роботи з таймером і портами введення-виведення (рис. 77).

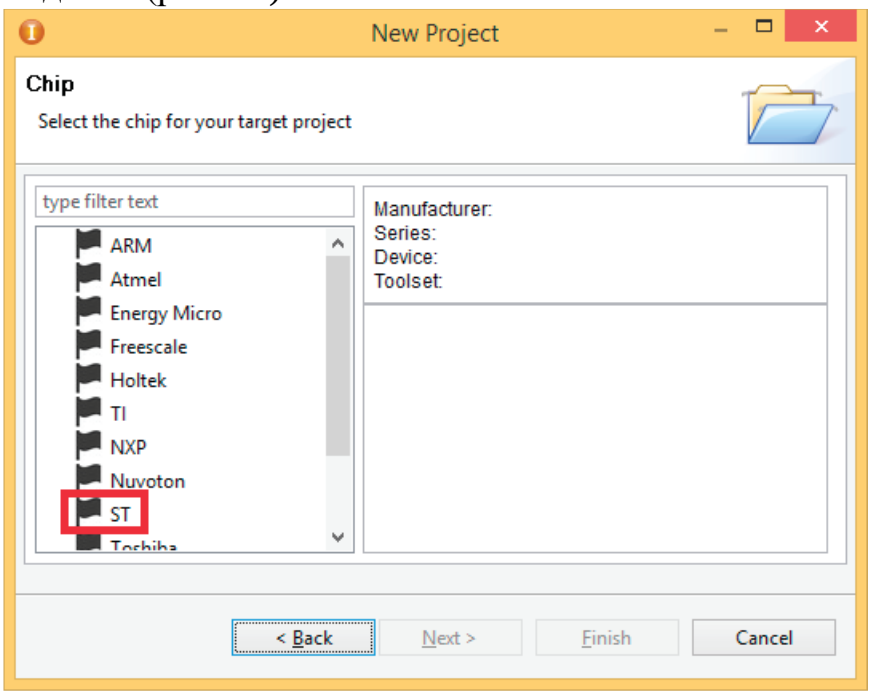


Рис. 73. Вибір фірми мікроконтролера

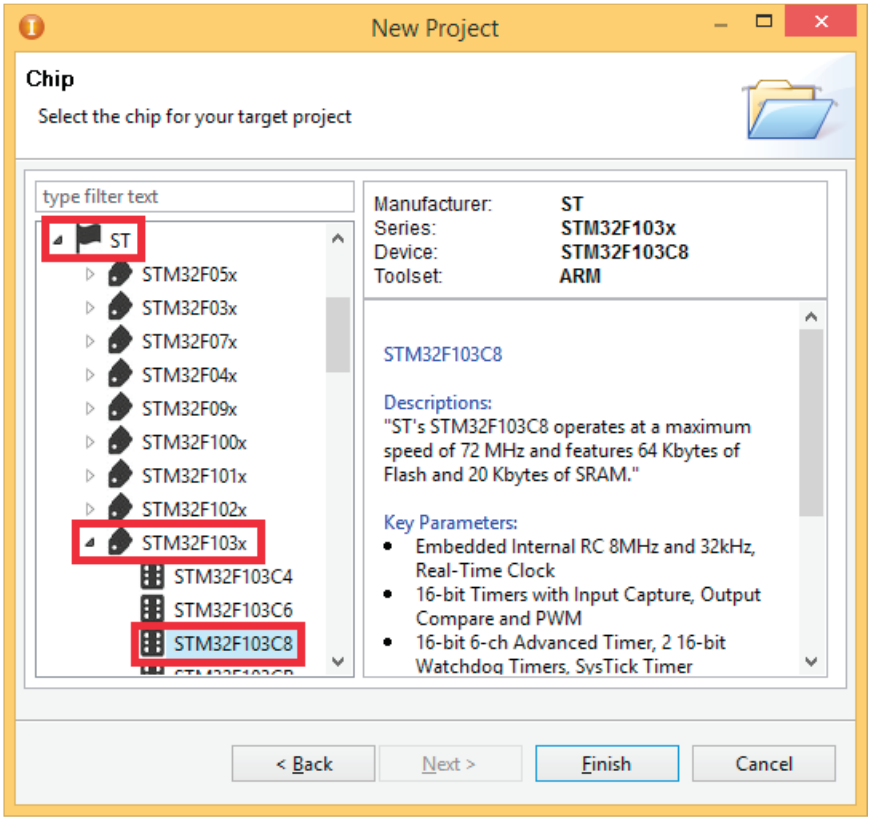


Рис. 74. Вибір мікроконтролера

9. Далі необхідно відкрити файл «timer.c». У цьому файлі потрібно створити функцію, в якій включити тактування периферійних пристроїв, а також ввести структури і заповнити їх для ініціалізації портів введення-виведення і таймера (TIM1). Зміст файлу «timer.c» представлено в лістингу 1.

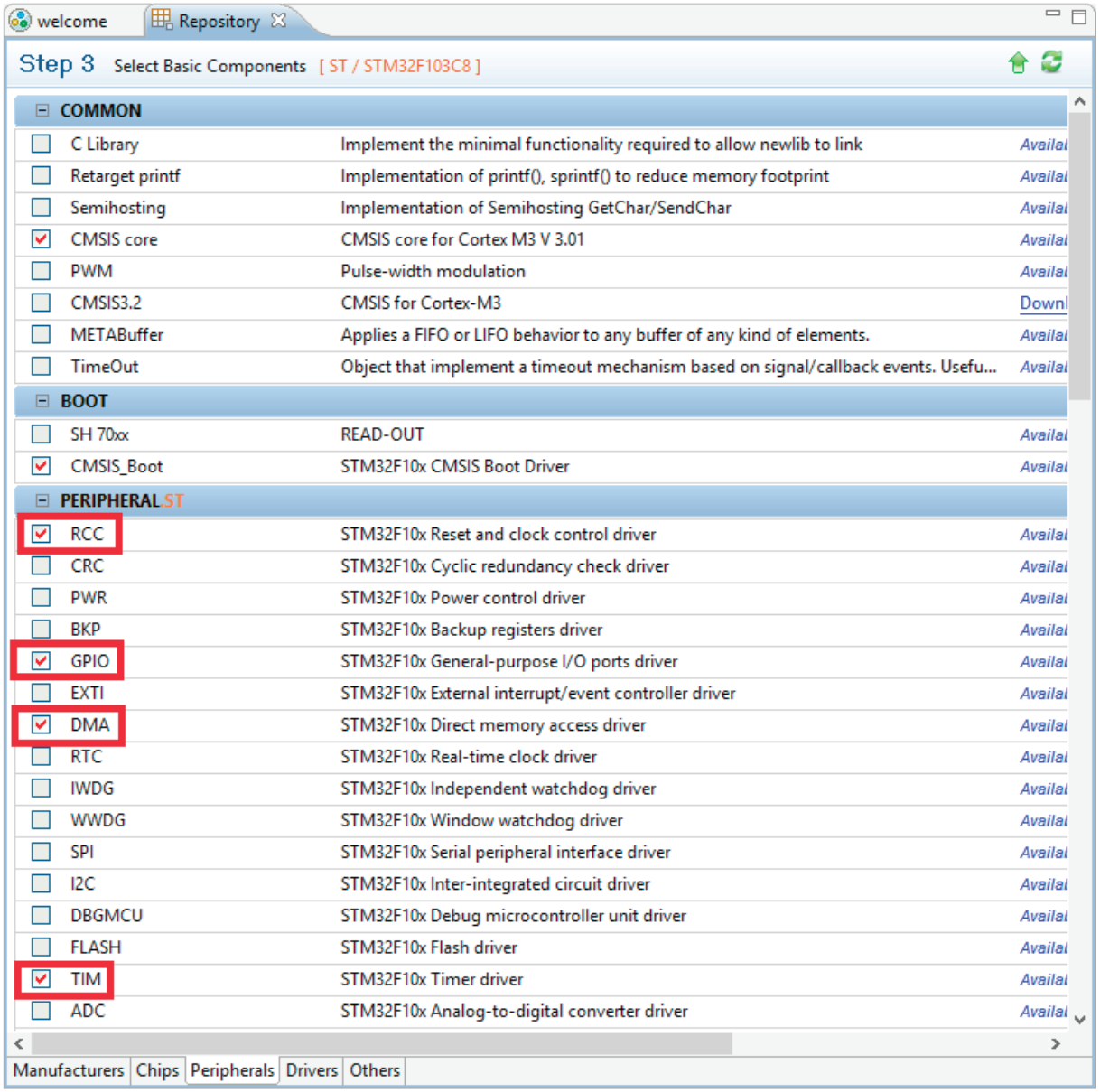


Рис. 75. Вибір бібліотек в репозиторії

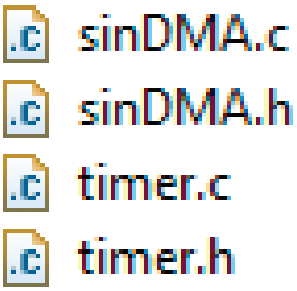


Рис. 76. Створення файлів «timer» і «sinDMA»

```
1 #include <stm32f10x.h>
2 #include <stm32f10x_gpio.h>
3 #include <stm32f10x_rcc.h>
4 #include <stm32f10x_tim.h>
```

Рис. 77. Запис заголовних файлів в «timer.h»

Лістинг 1. Зміст файлу «timer.c»:

```
# Include <timer.h>
void timers (void)
{
// ----- Включення тактирования ---- //
RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOA |
RCC_APB2Periph_AFIO
| RCC_APB2Periph_TIM1,ENABLE);
// ------ Введення структур ------- //
GPIO_InitTypeDef gpio;
GPIO_InitTypeDef gpio_1; GPIO_InitTypeDef
gpio_2; GPIO_InitTypeDef gpio_3;
TIM_TimeBaseInitTypeDef TIM_Base_1;
TIM_OCInitTypeDef TIM_PWM; // ---
Призначення GPIO ------ GPIO_StructInit (&
gpio);
gpio.GPIO_Mode = GPIO_Mode_AF_PP;
gpio.GPIO_Pin = GPIO_Pin_8;
gpio.GPIO_Speed =GPIO_Speed_50MHz;
GPIO_Init (GPIOA, & gpio); // ------------------
GPIO_StructInit (& gpio_1);
gpio_1.GPIO_Mode =GPIO_Mode_AF_PP;
gpio_1.GPIO_Pin = GPIO_Pin_9;
gpio_1.GPIO_Speed =GPIO_Speed_50MHz;
GPIO_Init (GPIOA, & gpio_1); // ------------------
GPIO_StructInit (& gpio_3);
gpio_3.GPIO_Mode =GPIO_Mode_AF_PP;
gpio_3.GPIO_Pin = GPIO_Pin_10;
gpio_3.GPIO_Speed =GPIO_Speed_50MHz;
GPIO_Init (GPIOA, & gpio_3); // ---
Призначення TIM1 --------
TIM_TimeBaseStructInit (& TIM_Base_1);
TIM_Base_1.TIM_Prescaler = 0;
TIM_Base_1.TIM_CounterMode = TIM_CounterMode_Up;
TIM_Base_1.TIM_Period = 2000;
```

TIM_Base_1.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInit (TIM1, & TIM_Base_1);
// ---- Призначення ШІМ --------- TIM_OCStructInit (&
TIM_PWM);
TIM_PWM.TIM_OCMode = TIM_OCMode_PWM1;
TIM_PWM.TIM_OutputState = TIM_OutputState_Enable;
TIM_PWM.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OC1Init (TIM1, & TIM_PWM);
TIM_OC1PreloadConfig (TIM1, TIM_OCPreload_Enable);
TIM_OC2Init (TIM1, & TIM_PWM);
TIM_OC2PreloadConfig (TIM1, TIM_OCPreload_Enable);
TIM_OC3Init (TIM1, & TIM_PWM);
TIM_OC3PreloadConfig (TIM1, TIM_OCPreload_Enable);
TIM_Cmd (TIM1,ENABLE);
TIM_CtrlPWMOutputs (TIM1, ENABLE);
}

10. Після цього в файлі «timer.h» записати прототип функції «timers ()» (рис. 78), а також за допомогою директиви «#Include <>» підключити заголовний файл «timer.h» в файлі «main.c».

```
5 void timers(void);
6
```

Рис. 78. Запис прототипу функції «timers ()»

11. Далі потрібно відкрити файл «sinDMA.h» і підключити заголовки, необхідні для роботи з DMA (рис. 79).

```
1 #include <stm32f10x_dma.h>
2 #include <stm32f10x_gpio.h>
3 #include <stm32f10x_rcc.h>
```

Рис. 79. Підключення заголовних файлів в «sinDMA.h»

12. Далі необхідно відкрити файл «sinDMA.c» і створити масиви для трьохфазної системи з синусоїдальними напруженнями, а також створити функції для передачі елементів масиву за допомогою DMA в регістри порівняння таймера (TIM1). Зміст файлу «sinDMA.c» представлено в лістингу 2.

Лістинг 2. Зміст файлу «sinDMA.c»:

 # Include <sinDMA.h> //
*- Фаза А * - //*
uint16_t sinA [192] = {56,84,113,141,169,197,225,253,281,309,337,365,392,
420,447,474,502,529,556,583,609,636,662,688,714,740,766,791,817,842,867,

891,916,940,964,988,1011,1035,1058,1080,1103,1125,1147,1169,1190,1211,
1232,1252,1272,1292,1312,1331,1350,1368,1386,1404,1422,1439,1456,1472,
1488,1504,1519,1534,1549,1563,1577,1590,1603,1616,1628,1640,1651,1662,
1673,1683,1693,1702,1711,1720,1728,1736,1743,1750,1756,1762,1768,1773,
1777,1782,1785,1789,1792,1794,1796,1798,1799,1799,1800,1799,1799,1798,
1796,1794,1792,1789,1785,1782,1777,1773,1768,1762,1756,1750,1743,1736,
1728,1720,1711,1702,1693,1683,1673,1662,1651,1640,1628,1616,1603,1590,
1577,1563,1549,1534,1519,1504,1488,1472,1456,1439,1422,1404,1386,1368,
1350,1331,1312,1292,1272,1252,1232,1211,1190,1169,1147,1125,1103,1080,
1058,1035,1011,988,964,940,916,891,867,842,817,791,766,740,714,688,662,
636,609,583,556,529,502,474,447,420,392,365,337,309,281,253,225,56}; /
********** //
*- Фаза B (+ 2 * pi / 3) - //*
uint16_t sinB [192] = {1529,1514,1499,1483,1467,1450,1433,1416,1398,1380,
1362,1343,1324,1305,1286,1266,1245,1225,1204,1183,1161,1140,1118,1095,
1073,1050,1027,1003,980,956,932,908,883,858,833,808,783,757,732,706,680,
653,627,600,574,547,520,493,465,438,411,383,355,328,300,272,244,216,188,
160,131,103,75,47,18,9,37,65,94,122,150,178,206,234,262,290,318,346,374,
401,429,456,484,511,538,565,591,618,645,671,697,723,749,774,800,825,850,
875,900,924,948,972,996,1019,1042,1065,1088,1110,1132,1154,1176,1197,
1218,1239,1259,1279,1299,1318,1337,1356,1374,1392,1410,1428,1445,1461,
1478,1494,1509,1524,1539,1568,1581,1595,1608,1620,1632,1644,1655,1666,
1677,1687,1696,1705,1714,1723,1731,1745,1752,1758,1764,1769,1774,1779,
1783,1786,1790,1792,1795,1797,1798,1799,1799,1799,1799,1798,1797,1795,
1793,1791,1788,1784,1780,1776,1771,1766,1760,1748,1741,1733,1725,1717,
1708,1699,1690,1680,1670,1659,1529}; /
********** **//
*- Фаза C (-2 * pi / 3) - //*
uint16_t sinC [192] = {1599,1612,1624,1636,1648,1659,1670,1680,1690,1699,
1708,1717,1725,1733,1741,1748,1754,1760,1766,1771,1776,1780,1784,1788,
1791,1793,1795,1797,1798,1799,1799,1799,1799,1798,1797,1795,1792,1790,
1786,1783,1779,1774,1769,1764,1758,1752,1745,1738,1731,1723,1714,1705,
1696,1687,1677,1666,1655,1644,1632,1620,1608,1595,1581,1568,1554,1539,
1524,1509,1494,1478,1461,1445,1428,1410,1392,1374,1356,1337,1318,1299,
1279,1259,1239,1218,1197,1176,1154,1132,1110,1088,1065,1042,1019,996,
972,948,924,900,875,850,825,800,774,749,723,697,671,645,618,591,565,538,
511,484,456,429,401,374,346,318,290,262,234,206,178,150,122,94,65,37,9,
18,47,75,103,131,160,188,216,244,272,300,328,355,383,411,438,465,493,
520,547,574,600,627,653,680,706,732,757,783,808,833,858,883,908,932,956,
980,1003,1027,1050,1073,1095,1118,1140,1161,1183,1204,1225,1245,1266,
1286,1305,1324,1343,1362,1380,1398,1416,1433,1586,1586}; /

```
********************************************** /
void sinDMA_PhaseA (void)
{
RCC_AHBPeriphClockCmd (RCC_AHBPeriph_DMA1, ENABLE);
DMA_InitTypeDef DMA_struct; DMA_StructInit (& DMA_struct);
DMA_struct.DMA_PeripheralBaseAddr = (uint32_t) & TIM1-> CCR1;
DMA_struct.DMA_MemoryBaseAddr = (uint32_t) & sinA [0];
DMA_struct.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_struct.DMA_BufferSize = 192; DMA_struct.DMA_PeripheralInc =
DMA_PeripheralInc_Disable; DMA_struct.DMA_MemoryInc =
DMA_MemoryInc_Enable; DMA_struct.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord; DMA_struct.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord; DMA_struct.DMA_Mode =
DMA_Mode_Circular;
DMA_Init (DMA1_Channel2, & DMA_struct);
DMA_Cmd (DMA1_Channel2,ENABLE);
TIM_DMACmd (TIM1, TIM_DMA_CC1, ENABLE);
}
void sinDMA_PhaseB (void)
{
DMA_InitTypeDef DMA_struct1; DMA_StructInit (& DMA_struct1);
DMA_struct1.DMA_PeripheralBaseAddr = (uint32_t) & TIM1-> CCR2;
DMA_struct1.DMA_MemoryBaseAddr = (uint32_t) & sinB [0];
DMA_struct1.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_struct1.DMA_BufferSize = 192; DMA_struct1.DMA_PeripheralInc =
DMA_PeripheralInc_Disable; DMA_struct1.DMA_MemoryInc =
DMA_MemoryInc_Enable; DMA_struct1.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord; DMA_struct1.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord; DMA_struct1.DMA_Mode =
DMA_Mode_Circular;
DMA_Init (DMA1_Channel3, & DMA_struct1);
DMA_Cmd (DMA1_Channel3,ENABLE);
TIM_DMACmd (TIM1, TIM_DMA_CC2, ENABLE);
}
void sinDMA_PhaseC (void)
{
DMA_InitTypeDef DMA_struct2; DMA_StructInit (& DMA_struct2);
DMA_struct2.DMA_PeripheralBaseAddr = (uint32_t) & TIM1-> CCR3;
DMA_struct2.DMA_MemoryBaseAddr = (uint32_t) & sinC [0];
DMA_struct2.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_struct2.DMA_BufferSize = 192; DMA_struct2.DMA_PeripheralInc =
```

DMA_PeripheralInc_Disable; DMA_struct2.DMA_MemoryInc = DMA_MemoryInc_Enable; DMA_struct2.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; DMA_struct2.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord; DMA_struct2.DMA_Mode = DMA_Mode_Circular;
DMA_Init (DMA1_Channel6, & DMA_struct2);
DMA_Cmd (DMA1_Channel6,ENABLE);
TIM_DMACmd (TIM1, TIM_DMA_CC3, ENABLE);
}

13. Далі необхідно записати прототипи функцій в файл «sinDMA.h» (рис. 80), а також підключити заголовки «sinDMA.h» в «main.c».

```
4 void sinDMA_PhaseA(void);
5 void sinDMA_PhaseB(void);
6 void sinDMA_PhaseC(void);
```

Рис. 80. Запис прототипів функцій для генерації синусоїдальних напруг

14. Після того, як всі файли були заповнені і підключені до «main.c», необхідно записати функції в основну функцію «main ()» (рис. 81). Цикл «while (1)» в даній програмі залишається порожнім.

```
1 #include <timer.h>
2 #include <sinDMA.h>
3 int main(void)
4 {
5 timers();
6 sinDMA_PhaseA();
7 sinDMA_PhaseB();
8 sinDMA_PhaseC();
9 while(1)
10     {
11
12 }
13 }
```

Рис. 82. Зміст функції «main ()»

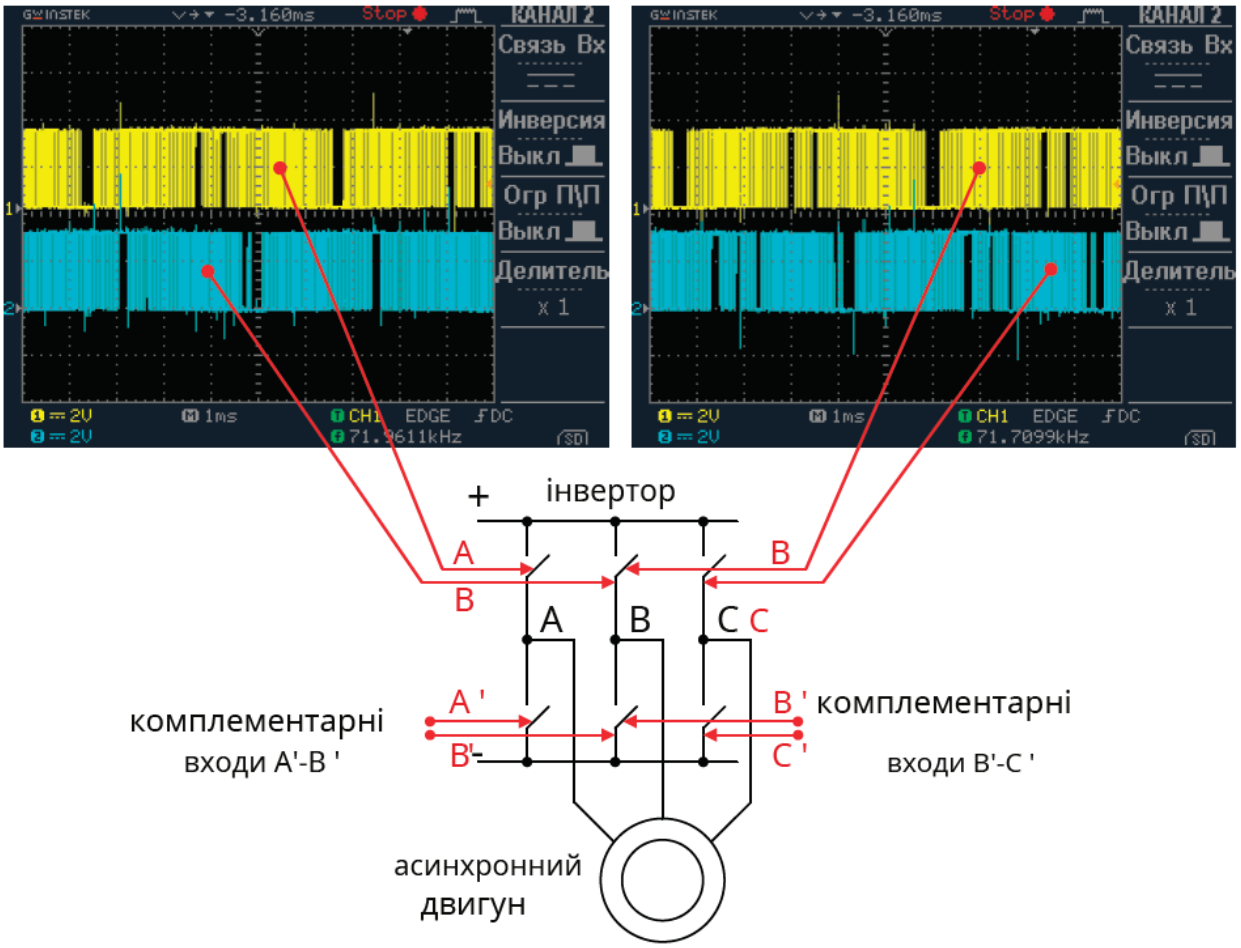


Рис. 83. Результати програмування сигналів AB і BC

15. Після написання коду програми, його необхідно скомпілювати. Для цього в панелі інструментів потрібно натиснути «Build». У разі успішної компіляції в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо ж в коді присутні помилки, то в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».
16. Після завершення компіляції останнім етапом стане завантаження робочої програми в мікроконтролер. Для цього потрібно через спеціальний кабель (подовжувач USB) підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження. У разі вдалої завантаження в консолі з'являться написи: «Erase: Done»; «Program: Done»; «Verify: Done». Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection». Необхідно перевірити кабель, до якого він підключений. Результати програмування трифазного генератора з синусоїдальними напругами із зсувом 120 ° дано на рис. 83.

Додаток Г

1. Запускаємо середовище програмування CooCox CoIDE.
2. Після запуску CooCox CoIDE в рядку меню натиснути: Project → New Project.
3. У вікні в полі «Project name» ввести ім'я свого проекту.
4. Далі потрібно вибрати поле з написом «Chip».
5. З'явиться вікно з випадаючими списками різних фірм мікроконтролерів (рис. 84). необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F103x, після чого знайти мікроконтролер STM32F103C8, вибрати його лівим клацанням миші і натиснути Finish (рис. 85).

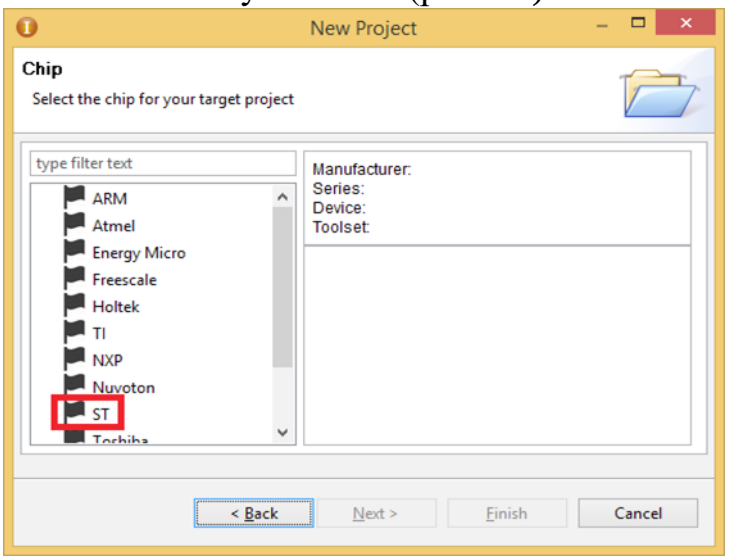


Рис. 84. Вибір фірми мікроконтролера

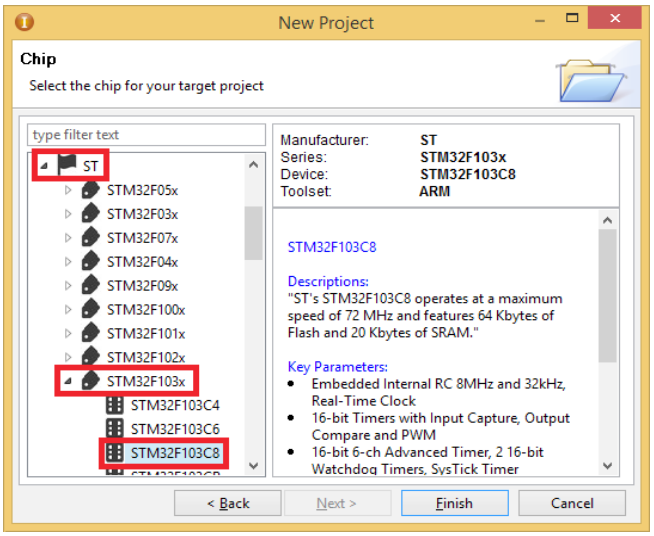


Рис. 85. Вибір мікроконтролера

6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору необхідних для проекту бібліотек (рис. 86). Необхідно підключити наступні бібліотеки:

-RCC-для управління тактовим генератором;
-GPIO-для управління портами введення-виведення;
-TIM-для управління таймерами;
-DMA-для роботи з прямим доступом до пам'яті;
-ADC-для роботи з аналогово-цифровим перетворювачем.

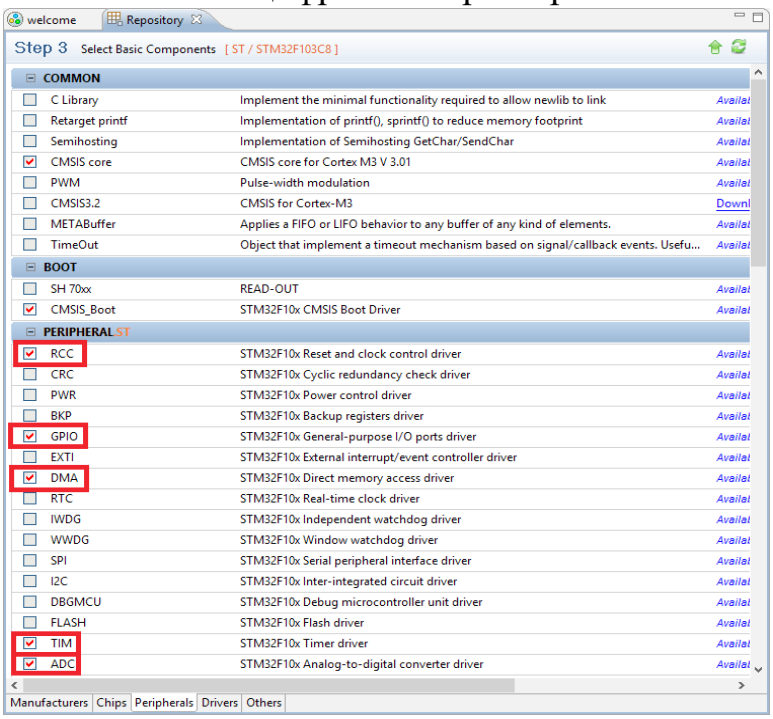


Рис. 86. Вибір бібліотек в репозиторії

7. Після вибору бібліотек необхідно на панелі інструментів вибрати «New file» і створити файли «ADC to DMA. h »і« main. h ». Потім створити файли «SetToRcc72» і «timerPWM» з розширенням «.з», а також аналогічні файли з розширенням «.h» (рис. 87).

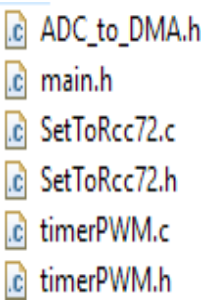


Рис. 87. Створення файлів проекту

8. Наступним кроком необхідно в панелі файлів відкрити «SetToRcc72. h », два рази клацнувши по ньому лівою кнопкою миші, і за допомогою

директиви «#include <>« записати в ньому заголовки «stm32f10x_rcc. h », необхідний для роботи з тактовим генератором (рис. 88).

```
1 #include "stm32f10x_rcc.h"
2
```

Рис. 88. Запис заголовків «stm32f10x_rcc. h »в« SetToRcc72. h »

9. Далі необхідно відкрити файл «SetToRcc72. c »і заповнити його. У ньому вказуються настройки тактирования мікроконтролера, значення множника частоти (PLL), а також подільники на шинах мікроконтролера (AHB, APB1, APB2). Зміст файлу «SetToRcc72. c »представлено в лістингу 1.
Лістинг 1. Зміст файлу «SetToRcc72. c »:

```
# Include "SetToRcc72.h" void rcc_pll (void)
{
ErrorStatus HSEStartUpStatus; // - Скидання налаштувань тактирования -
// RCC_DeInit ();

// - Включення зовнішнього кварцу на 8 МГц//
RCC_HSEConfig (RCC_HSE_ON); HSEStartUpStatus =
RCC_WaitForHSEStartUp (); if (HSEStartUpStatus == SUCCESS)
{
// --- Переддільник на шині AHB // RCC_HCLKConfig
(RCC_SYSCLK_Div1); // Переддільник на шині APB2 - //
RCC_PCLK2Config (RCC_HCLK_Div4); // Переддільник на шині
APB1 - // RCC_PCLK1Config (RCC_HCLK_Div2); // - Налаштування
тактирования для АЦП // RCC_ADCCLKConfig
(RCC_PCLK2_Div2); // - Множення частоти на 9 //
RCC_PLLConfig (RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); // -
Включення множення //


RCC_PLLCmd (ENABLE);
while (RCC_GetFlagStatus (RCC_FLAG_PLLRDY) == RESET)
{
}
RCC_SYSCLKConfig (RCC_SYSCLKSource_PLLCLK); while
(RCC_GetSYSCLKSource ()! = 0x08)
{
```

}
}
else { while (1)
{
}
}
}

10. Після написання коду необхідно записати прототип функції в файл «SetToRcc72. h » (рис. 89).

```
2 void rcc_pll(void);
3
```

Рис. 89. Прототип функції у файлі «SetToRcc72. h »

11. Наступним кроком необхідно відкрити файл «main. h »і підключити туди все заголовки (рис. 90), котрі будуть використовуватися в проекті. Це дозволить до всіх файлів підключати тільки «main. h ».

```
1 #include "stm32f10x.h"
2 #include "stm32f10x_rcc.h"
3 #include "stm32f10x_tim.h"
4 #include "stm32f10x_gpio.h"
5 #include "stm32f10x_dma.h"
6 #include "stm32f10x_adc.h"
```

Рис. 90. Підключення заголовних файлів в «main. h »

12. Далі необхідно відкрити файл «ADC to DMA. h »і за допомогою директиви« #include <> «підключити до нього файл «main. h ». Потім потрібно створити масив з ключовим словом «volatile» (повідомляє компілятору, що зміст масиву може бути змінено ззовні), в який будуть записуватися значення з потенціометра, а також створити функцію для ініціалізації DMA і АЦП і заповнити її. Зміст файлу «ADC to DMA. h »представлено в лістингу 2.

Лістинг 2. Зміст файлу «ADC to DMA. h »:

 # Include «main. h »
// - -Маса для збереження даних з потенціометра - - // volatile uint16_t ADCBuffer [] = {0xAAA};
void adc_to_dma (void)
{

// - Налаштування DMA для роботи з регістром АЦП - - // RCC_AHBPeriphClockCmd (RCC_AHBPeriph_DMA1, ENABLE); DMA_InitTypeDef DMA_Struct;
DMA_Struct. DMA_PeripheralBaseAddr = (uint32_t) & ADC1-> DR; DMA_Struct. DMA_MemoryBaseAddr = (uint32_t) ADCBuffer; DMA_Struct. DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_Struct. DMA_BufferSize = 1;
DMA_Struct. DMA_PeripheralInc = DMA_PeripheralInc_Disable; DMA_Struct. DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_Struct. DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; DMA_Struct. DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord; DMA_Struct. DMA_Mode = DMA_Mode_Circular;
DMA_Struct. DMA_Priority = DMA_Priority_Medium; DMA_Struct. DMA_M2M = DMA_M2M_Disable; DMA_Init (DMA1_Channel1, & DMA_Struct); DMA_Cmd (DMA1_Channel1, ENABLE);
// - -Настройка аналогового входу - - //

RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOB, ENABLE); GPIO_InitTypeDef poten;
poten. GPIO_Mode = GPIO_Mode_AIN; poten. GPIO_Pin = GPIO_Pin_1;
poten. GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init (GPIOB, & poten);
*/ * - Налаштування АЦП для роботи з одним каналом і передачі даних по DMA - * / RCC_APB2PeriphClockCmd (RCC_APB2Periph_ADC1, ENABLE); ADC_InitTypeDef adc_user;*
adc_user. ADC_Mode = ADC_Mode_Independent; adc_user. ADC_ScanConvMode = DISABLE;
adc_user. ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; adc_user. ADC_DataAlign = ADC_DataAlign_Right;
adc_user. ADC_NbrOfChannel = 1; ADC_Init (ADC1, & adc_user);
ADC_RegularChannelConfig (ADC1, ADC_Channel_9,1, ADC_SampleTime_55Cycles5); ADC_Cmd (ADC1, ENABLE);
ADC_DMACmd (ADC1, ENABLE); // - Калібрування АЦП - // ADC_ResetCalibration (ADC1);
while (ADC_GetResetCalibrationStatus (ADC1)); ADC_StartCalibration (ADC1);
while (ADC_GetCalibrationStatus (ADC1)); ADC_SoftwareStartConvCmd (ADC1, ENABLE); }

13. Далі необхідно відкрити файл «timerPWM. h »і за допомогою директиви« #include <> «підключити до нього файл «main. h».
14. Далі потрібно відкрити файл «timerPWM. c» і підключити до нього заголовки «main. h». У цьому файлі розташовуватимуться дві функції. Перша функція буде використовуватися для ініціалізації таймера (TIM1) і налаштування першого ШІМ-каналу. Друга функція буде приймати в аргументі значення з потенціометра і відправляти його в регістр порівняння першого каналу таймера (TIM1). Також тут знаходиться функція для запуску перетворення вхідних даних АЦП. Зміст файлу «timerPWM. c » представлено в лістингу 3.

Лістинг 3. Зміст файлу «timerPWM. c »:

```
# Include «timerPWM. h
»void timerPWM (void)
{
// - Включення тактирования - - // RCC_APB2PeriphClockCmd
(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA, ENABLE);
// - -Настройка виведення як альтернативної функції - //
GPIO_InitTypeDef gpio;
gpio. GPIO_Mode = GPIO_Mode_AF_PP; gpio. GPIO_Pin = GPIO_Pin_8;
gpio. GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init (GPIOA, & gpio);
// - -Настройка виведення як альтернативної функції - //
GPIO_InitTypeDef gpio_1;
GPIO_StructInit (& gpio_1);
gpio_1. GPIO_Mode = GPIO_Mode_AF_PP; gpio_1. GPIO_Pin =
GPIO_Pin_13; gpio_1. GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init
(GPIOB, & gpio_1);

// - Налаштування таймера TIM1- - - // RCC_APB2PeriphClockCmd
(RCC_APB2Periph_TIM1, ENABLE); TIM_TimeBaseInitTypeDef
timer_TIM1;
TIM_TimeBaseStructInit (& timer_TIM1);

timer_TIM1. TIM_Prescaler = 0;
timer_TIM1. TIM_CounterMode = TIM_CounterMode_Up; timer_TIM1.
TIM_Period = 5000;
timer_TIM1. TIM_ClockDivision = 0; timer_TIM1. TIM_RepetitionCounter =
0; TIM_TimeBaseInit (TIM1, & timer_TIM1); // - Налаштування ШІМ - - //

TIM_OCInitTypeDef timer_TIM1_OC; TIM_BDTRInitTypeDef bdtr_struct;
TIM_OCStructInit (& timer_TIM1_OC); timer_TIM1_OC. TIM_OCMode =
```

TIM_OCMode_PWM1; timer_TIM1_OC. TIM_OutputState = TIM_OutputState_Enable; timer_TIM1_OC. TIM_OutputNState = TIM_OutputNState_Enable; TIM_OC1Init (TIM1, & timer_TIM1_OC); // - Налаштування мертвого часу - - // TIM_BDTRStructInit (& bdtr_struct);
bdtr_struct. TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;
*bdtr_struct. TIM_DeadTime = 36 * 6;*
TIM_BDTRConfig (TIM1, & bdtr_struct); // - Включення таймера і ШІМ - - // TIM_Cmd (TIM1, ENABLE);
TIM_CtrlPWMOutputs (TIM1, ENABLE); }
// - Функція запису даних в регістри порівняння - // void SetPWM (uint16_t PWM)
{
ADC_SoftwareStartConvCmd (ADC1, ENABLE); TIM1-> CCR1 = PWM;
}

15. Після написання коду необхідно в файлі «timerPWM. h» вказати прототипи функцій (рис. 91).

```
2
3 void timerPWM(void);
4 void SetPWM(uint16_t PWM);
5
```

Рис. 91. Прототипи функцій в файлі «timerPWM. h»

16. Наступним кроком необхідно відкрити файл «main. c», підключити всі використовувані заголовки та додати всі використовувані функції в основну функцію «main ()». У нескінченний цикл «while (1)» необхідно написати функцію запису даних в регістри порівняння (рис. 92).

```
1 #include <main.h>
2 #include "ADC_to_DMA.h"
3 #include "SetToRcc72.h"
4 #include "timerPWM.h"
5 int main(void)
6 {
7 rcc_pll();
8 adc_to_dma();
9 timerPWM();
10     while(1)
11     {
12 SetPWM(ADCBuffer[0]);
13     }
14 }
```

Рис. 92. Підключення заголовних файлів і використовуваних функцій в «main. c»

17. Після написання коду програми, його необхідно скомпілювати. Для цього в панелі інструментів потрібно натиснути «Build». У разі успішної компіляції в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо в коді присутні помилки, то в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».

18. Після завершення компіляції останнім етапом стане завантаження робочої програми в мікроконтролер. для цього потрібно через спеціальний кабель (подовжувач USB) підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження. У разі вдалої завантаження в консолі з'являться написи: «Erase: Done»; «Program: Done»; «Verify: Done». Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection». Необхідно перевірити кабель, до якого він підключений.

Результати зміни шпаруватості імпульсів ШІМ при різних положеннях потенціометра дані на рис. 93.

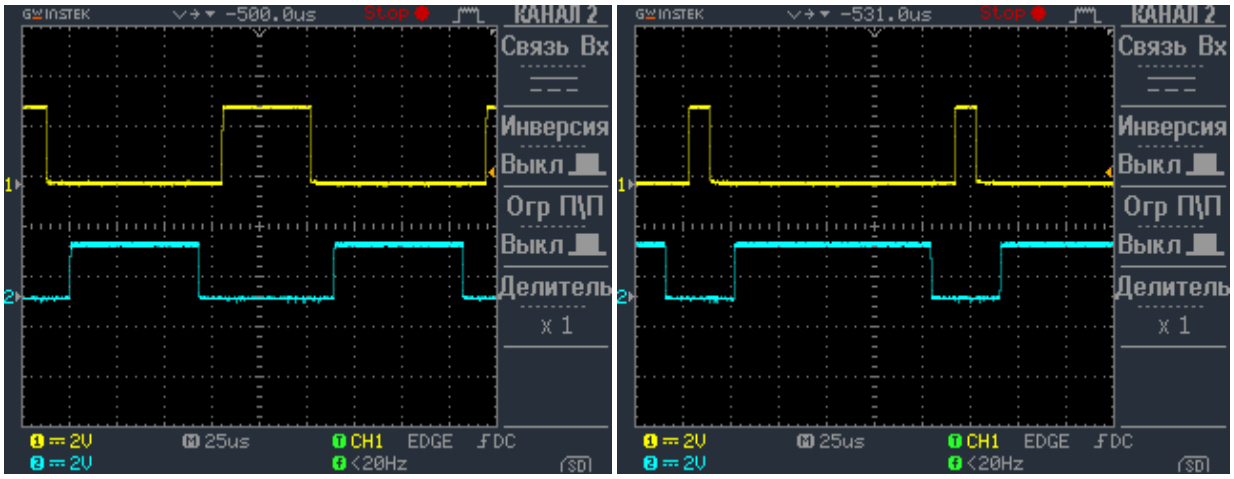


Рис. 93. Осцилограми зміни шпаруватості сигналу при різних положеннях потенціометра з налаштуванням «мертвого часу»

Додаток Д

1. Запускаємо середовище програмування CooCox CoIDE.
2. Після запуску CooCox CoIDE в рядку меню натиснути: Project →New Project.
3. У вікні в полі «Project name» ввести ім'я свого проекту.
4. Далі потрібно вибрати поле з написом «Chip».
5. З'явиться вікно з випадаючими списками різних фірм мікроконтролерів. Необхідно відкрити список ST, потім зі списку відкрити підсписок STM32F4x, після чого знайти мікроконтролер STM32F407VG, вибрати його лівим клацанням миші і натиснути Finish.
6. Після виконаних дій з'явиться головне вікно з репозиторієм для вибору бібліотек. Необхідно підключити наступні бібліотеки:
- RCC-для управління тактовим генератором;
- GPIO-для управління портами введення-виведення;
- TIM-для управління таймерами;
- DMA-для керування прямим доступом до пам'яті (DMA);
- DAC-для роботи з цифро-аналоговим перетворювачем.
7. Після вибору бібліотек необхідно на панелі інструментів вибрати «New file» і створити файли «dac_user.c» і «dac_user.h» (рис. 94).

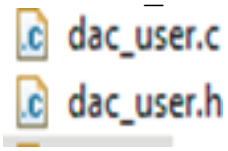


Рис. 94. Створення файлів «dac_user.c» і «dac_user.h»

1. У панелі файлів відкрити «dac_user.h», два рази клацнувши по ньому лівою кнопкою миші, і з допомогою директиви «#include <>» записати в ньому всі заголовки, необхідні для роботи з таймером (TIM), портами введення-виведення (GPIO), прямим доступом до пам'яті (DMA) і ЦАП (DAC) (рис. 95).

```
1 #include <stm32f4xx_dac.h>
2 #include <stm32f4xx_gpio.h>
3 #include <stm32f4xx_rcc.h>
4 #include <stm32f4xx_tim.h>
5 #include <stm32f4xx_dma.h>
6 #include <stm32f4xx.h>
```

Рис. 95. Запис заголовних файлів

2. Відкрити файл «dac_user.c» і створити в ньому функцію, в якій необхідно включити тактування периферійних пристроїв, а також ввести структури і заповнити їх для ініціалізації портів введення-виведення, таймера (TIM4), DMA і ЦАП. Зміст файлу «dac_user.c» представлено в лістингу 1.

Лістинг 1. Зміст файлу «dac_user.c»:

```
# include «Dac_user.h»
void dac_triangle_sin (void)
{
// - Таблиця значень синусоїди - //
uint16_t sin [74] = {2225, 2402, 2577, 2747,2912,3070,3221,3363,3494,3615,
3724,3820,3902,3971,4024,4063,4085,4095,4063,4024,3971,3902,3820,3724,
3615,3495,3363,3221,3071,2912,2747,2577,2403,2226,2047,1869,1692,1517,
1347,1182,1024,873,731,600,479,370,274,192,274,124,70,31,31,10,0,10,31,
70,123,192,274,370,479,599,599,731,873,1023,1182,1347,1517,1691,1868,



2047};
// - Тактирование периферійних пристроїв - // RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_DMA1, ENABLE); RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM4 | RCC_APB1Periph_DAC, ENABLE);
// - Ініціалізація 4 виведення GPIO - // GPIO_InitTypeDef gpio_dacchannel1; GPIO_StructInit (& gpio_dacchannel1); gpio_dacchannel1.GPIO_Mode =GPIO_Mode_AF; gpio_dacchannel1.GPIO_OType = GPIO_OType_PP; gpio_dacchannel1.GPIO_Pin = GPIO_Pin_4; gpio_dacchannel1.GPIO_PuPd =GPIO_PuPd_UP;
gpio_dacchannel1.GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init (GPIOA, & gpio_dacchannel1); // - Ініціалізація 5 виведення GPIO - //
GPIO_InitTypeDef gpio_dacchannel2; GPIO_StructInit (& gpio_dacchannel2); gpio_dacchannel2.GPIO_Mode =GPIO_Mode_AF; gpio_dacchannel2.GPIO_OType = GPIO_OType_PP; gpio_dacchannel2.GPIO_Pin = GPIO_Pin_5; gpio_dacchannel2.GPIO_PuPd =GPIO_PuPd_UP;
gpio_dacchannel2.GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init (GPIOA, & gpio_dacchannel2); // - Ініціалізація таймера TIM4 як тригера - //
TIM_TimeBaseInitTypeDef timer;
TIM_TimeBaseStructInit (& timer); timer.TIM_ClockDivision = TIM_CKD_DIV1;
timer.TIM_CounterMode = TIM_CounterMode_Up; timer.TIM_Period = 20-1;
timer.TIM_Prescaler = 10; TIM_TimeBaseInit (TIM4, & timer);
TIM_SelectOutputTrigger (TIM4, TIM_TRGOSource_Update); // - Ініціалізація альтернативних функцій - // GPIO_PinAFConfig (GPIOA, GPIO_PinSource4, GPIO_AF_TIM4); GPIO_PinAFConfig (GPIOA, GPIO_PinSource5, GPIO_AF_TIM4); // - Ініціалізація ЦАП, канал 1 - //
```

```
DAC_InitTypeDef dac_triangle; DAC_StructInit (& dac_triangle);
dac_triangle.DAC_Trigger = DAC_Trigger_T4_TRGO;
dac_triangle.DAC_WaveGeneration
= DAC_WaveGeneration_Triangle;
dac_triangle.DAC_LFSRUnmask_TriangleAmplitude =
DAC_TriangleAmplitude_4095; dac_triangle.DAC_OutputBuffer =
DAC_OutputBuffer_Enable;

DAC_Init (DAC_Channel_1, & dac_triangle); DAC_Cmd
(DAC_Channel_1,ENABLE);
// - Ініціалізація DMA - // DMA_InitTypeDef DMA_dac; DMA_StructInit
(& DMA_dac); DMA_dac.DMA_Channel = DMA_Channel_7;
DMA_dac.DMA_PeripheralBaseAddr = (uint32_t) & (DAC-> DHR12R2);
DMA_dac.DMA_Memory0BaseAddr = (uint32_t) sin;

DMA_dac.DMA_DIR = DMA_DIR_MemoryToPeripheral;
DMA_dac.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_dac.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_dac.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_dac.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_dac.DMA_Mode = DMA_Mode_Circular;
DMA_dac.DMA_Priority = DMA_Priority_Medium;


DMA_dac.DMA_FIFOMode = DMA_FIFOMode_Disable;
DMA_dac.DMA_FIFOThreshold = DMA_FIFOThreshold_1QuarterFull;
DMA_dac.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_dac.DMA_PeripheralBurst = DMA_PeripheralBurst_Single; DMA_Init
(DMA1_Stream6, & DMA_dac);
// - Ініціалізація ЦАП, канал 2 - //
DAC_InitTypeDef dac_sin;
DAC_StructInit (& dac_sin); dac_sin.DAC_Trigger = DAC_Trigger_T4_TRGO;
dac_sin.DAC_WaveGeneration = DAC_WaveGeneration_None;
dac_sin.DAC_OutputBuffer = DAC_OutputBuffer_Enable; DAC_Init
(DAC_Channel_2, & dac_sin);
DAC_DMACmd (DAC_Channel_2, ENABLE);
DAC_Cmd (DAC_Channel_2, ENABLE); DMA_Cmd (DMA1_Stream6, ENABLE);
TIM_Cmd (TIM4, ENABLE);
}
```

3. Після цього в файлі «dac_user.h» записати прототип функції «dac_triangle_sin ()» (рис. 96), а також за допомогою директиви «#include <>» підключити заголовковий файл «dac_user.h» в «main.c».

```
7 void dac_triangle_sin(void);
8
```

Рис. 96. Запис прототипу функції «dac_triangle_sin ()»

4. Після того, як всі файли були заповнені і підключені до «main.c», необхідно записати функцію в основну функцію «main ()» (рис. 97). Цикл «while (1)» в даній програмі залишається порожнім.

```
1 #include "dac_user.h"
2 int main(void)
3 {
4 dac_triangle_sin();
5
6     while(1)
7     {
8
9     }
.0 }
```

Рис. 97. Зміст функції «main ()»

5. Після написання коду програми, його необхідно скомпілювати. Для цього в панелі інструментів потрібно натиснути «Build». У разі успішної компіляції в консолі з'явиться напис «BUILD SUCCESSFUL», а також буде вказано розмір програми. Якщо в коді присутні помилки, то в консолі буде вказано, де саме знаходяться ці помилки, а також з'явиться напис «BUILD FAILED».

6. Після завершення компіляції останнім етапом стане завантаження робочої програми в мікроконтролер. Для цього потрібно через спеціальний кабель (подовжувач USB) підключити програматор, розташований на лабораторному стенді, до комп'ютера. Після підключення в панелі інструментів натиснути «Download Code to Flash» і дочекатися закінчення завантаження. У разі вдалої завантаження в консолі з'являться написи: «Erase: Done»; «Program: Done»; «Verify: Done». Якщо існують проблеми з підключенням плати до комп'ютера, то з'явиться напис «Error: Connect failed, check config and cable connection». Необхідно перевірити кабель, до якого він підключений.

Результати програмування синусоїдального і пилоподібного сигналів за допомогою цифро-аналогового перетворювача дані на рис. 98.

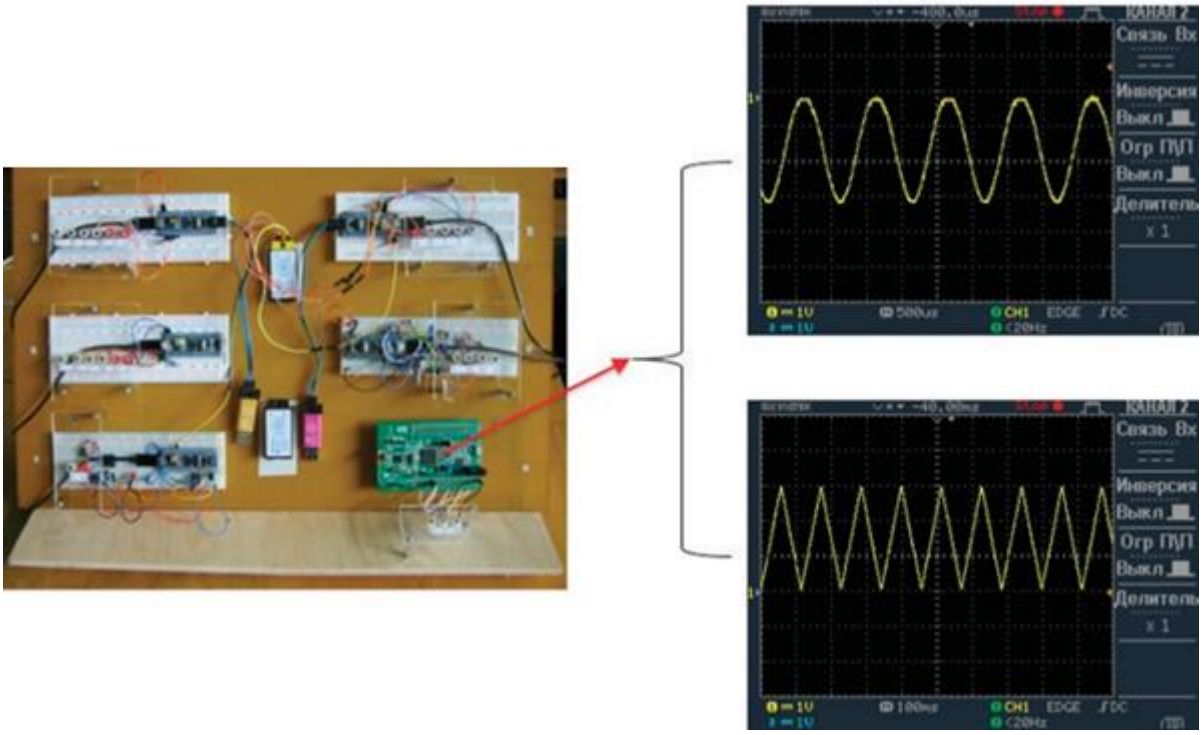


Рис. 98. Лабораторний стенд для дослідження генерації пилоподібного і синусоїдального сигналів за допомогою ЦАП