Міністерство освіти і науки України
Донбаська державна машинобудівна академія (ДДМА)

В. О. Квашнін,

А. В. Бабаш,

В. В. Квашнін

ПРОГРАМУВАННЯ ТА ЗАСТОСУВАННЯ МІКРОКОНТРОЛЕРІВ STM32F4DISCOVERY

Монографія

Затверджено
на засіданні вченої ради
Протокол № 8 від 27.04.2017

Краматорськ
ЦТРІ «Друкарський дім»
2017

УДК 62.83
К 32

Рецензенти:

Волков І. В., д-р техн. наук, професор, член-кореспондент НАН України, Інститут електродинаміки НАН України (м. Київ);

Тарасов А. Ф., д-р техн. наук, професор, завідувач кафедри комп'ютерних інформаційних технологій, Донбаська державна машинобудівна академія (м. Краматорськ);

Стяжкін В. П., канд. техн. наук, старший науковий співробітник відділу систем стабілізованого струму, Інститут електродинаміки НАН України (м. Київ).

Видання здійснено за підтримки міжнародного проекту «Розробка курсів із вбудованих систем з використанням інноваційних віртуальних підходів для інтеграції науки, освіти та промисловості в Україні, Грузії та Вірменії DesIRE» (544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR) за програмою TEMPUS Європейської комісії.

Квашнін, В. О.

К 32 Програмування та застосування мікроконтролерів STM32F4Discovery : монографія / В. О. Квашнін, А. В. Бабаш, В. В. Квашнін. – Краматорськ : ЦТРІ «Друкарський дім», 2017. – 143 с.

ISBN 978-617-7415-30-4.

Містить опис 32-розрядних мікроконтролерів STM32F4, їхні основні характеристики. Наведено основні конструкції мови програмування С, використовуваної для програмування мікроконтролерів STM32F4, опис налаштування портів введення-виведення, алгоритм налаштування середовища програмування для мікроконтролерів.

Подано практичні приклади з налаштування й роботи з навчальною платою. Монографія розрахована на підготовку бакалаврів і магістрів комп'ютерних і електротехнічніх спеціальностей, а також на широке коло користувачів, що розробляють різні мікропроцесорні пристрої на основі програмованих мікроконтролерів STM32F4.

УДК 62.83

ISBN 978-617-7415-30-4

ЗМІСТ

ВСТУП

При інтеграції вітчизняної системи освіти в загальноєвропейську систему підготовка фахівців повинна бути зорієнтована на зростаючі вимоги, що ставляться до них. Зокрема, вона повинна здійснюватися на основі використання сучасних зразків високоінтелектуального автоматизованого обладнання, яке повинно будуватися з окремих технологічних модулів. Подібні технологічні модулі повинні бути функціонально закінченими, автономно функціонуючими й наділеними властивостями бути інтегрованими в більш високі рівні системи керування.

У нинішніх важких економічних умовах навчальні заклади не мають можливостей для суттєвого оновлення матеріальної навчальної та лабораторної бази. Єдиним виходом в умовах, що склалися, є участь в різних міжнародних проектах і отримання грантів. Зокрема, за підтримки Tempus DesIRE project на факультеті ФАМІТ у цілому, і на кафедрі ЕСА в тому числі, з'явилося сучасне обладнання у вигляді мікроконтролерів Arduino і STM32F4, а також одноплатні комп'ютери Raspberry PI.

Слід зазначити, що використання мікроконтролерів STM32F100 на кафедрі почалося ще до участі в зазначеній міжнародній програмі. Тому при опануванні нової серії мікроконтролерів STM32F4 був використаний досвід, напрацьований при роботі з мікроконтролерами STM32F100 спрощеної версії. Наявність сучасних мікроконтролерів серії STM32F4 дає можливість проводити роботи з розроблення й проектування сучасних частотних електроприводів, вирішувати завдання високоточного вимірювання й діагностування їх параметрів, а також будувати сучасні системи контролю та керування на їх основі. Однак відсутність відповідного програмного забезпечення й відповідної підготовки обслуговчого персоналу стримує широке їх використання в навчальному процесі для підготовки бакалаврів і магістрів. Наявний опис даних мікроконтролерів [1] орієнтований або на вже заздалегідь підготовлених фахівців, або просто містить набір технічних характеристик і перелік спеціалізованих функцій без їх розкриття.

На основі вищевикладеного виникає необхідність у розробленні докладного технічного опису цих мікроконтролерів з деталізацією їх практичного застосування у вигляді конкретних прикладів, розрахованих на непідготовлений персонал, у якому буде враховано досвід їх практичного застосування з поданням відповідного програмного забезпечення з докладним його описом. Для досягнення поставленої мети необхідно визначити можливі області його практичного застосування, розробити детальний опис технічних можливостей мікроконтролера STM32F4 щодо його практичного застосування, розробити відповідний протокол передавання даних для зв'язку мікроконтролера й персонального комп'ютера, розробити відповідне програмне забезпечення для підтримання й роботи цифро-аналогового (ЦАП), аналого-цифрового (АЦП) перетворювачів, формування сигналів за допомогою широтно-імпульсного модулювання (ШІМ) тощо.

Крім того, для роботи мікроконтролера STM32F4 необхідно було визначити середу програмування. Вибір був зроблений на користь вільного середовища Atollic True Studio, яке підтримує мови програмування високого рівня, такі як C і C ++ (проте можна використовувати середовище програмування Keil). Для програмування мікроконтролера STM32F4 використовується мова високого рівня C. Мова C дозволяє працювати з мікроконтролером як на високому рівні (з використанням спеціалізованих бібліотек і функцій), так і на низькому рівні (можна безпосередньо працювати з бітами спеціалізованих регістрів мікроконтролера). Необхідне було також розроблення короткого, але ємного опису мови програмування C, розрахованого на непідготовленого користувача.

Монографія орієнтована на самостійне вивчення можливостей 32-розрядних мікроконтролерів STM32F4, їх використання в побудові інформаційних і керівних систем, організації взаємодії декількох пристроїв між собою, використання, передавання й відображення інформації в комп'ютерних системах, навчання самостійної розробки програмного забезпечення з використанням спеціалізованих програм, проведення тестування та налаштування на реальних пристроях.

При розгляді практичних прикладів використання або застосування мікроконтролера розглядаються принципи його роботи, периферії, організації передавання даних з його використанням, керування іншими пристроями для вимірювання зовнішніх параметрів, налаштування відображення інформації, отриманої від зовнішніх пристроїв. Монографія допомагає самостійно налаштувати роботу демонстраційних прикладів і розробити власні відповідно до запропонованих можливих варіантів практичної реалізації.

Етапи практичного використання налагоджувальної плати STM32F4Discovery включають у себе такі основні розділи:

- загальний опис архітектури ARM і 32-розрядних мікроконтролерів STM;
- загальна інформація, необхідна для початку роботи з налагоджувальною платою STM32F4Discovery;
- опис мови програмування C, яка використовується для написання програм для мікроконтролерів STM32;
- опис підключення налагоджувальної плати до комп'ютера (необхідні апаратні засоби, програмне забезпечення, яке потрібно встановити для нормальної роботи плати);
- перелік практичних робіт для вивчення основних можливостей, пристроїв і характеристик плати: ШІМ, АЦП, ЦАП, USART таймерів тощо.

Для виконання практичних прикладів бажане знання основ теорії цифрової схемотехніки, розроблення програмного забезпечення та алгоритмізації, а також знання мови програмування C.

РОЗДІЛ 1
ЗНАЙОМСТВО ІЗ СУЧАСНИМИ МІКРОПРОЦЕСОРНИМИ ПРИСТРОЯМИ

1.1 Архітектура ARM

Архітектура ARM (Advanced RISC Machine, Acorn RISC Machine, вдосконалена RISC-машина) – сімейство ліцензованих 32- і 64-бітових мікропроцесорних ядер розроблення компанії ARM Limited.

Серед ліцензіатів: AMD, Apple, Analog Devices, Atmel, Xilinx, Altera, Cirrus Logic, Intel (до 27 червня 2006 року), Marvell, NXP, STMicroelectronics, Samsung, LG, MediaTek, MStar, Qualcomm, Sony, Texas Instruments, nVidia, Freescale, Міландр, HiSilicon.

1.2 Процесори ARM

У цей час значущими є кілька сімейств процесорів ARM:

- ARM7 (з тактовою частотою до 60…72 МГц) призначені для недорогих мобільних телефонів і вбудованих рішень середньої продуктивності. Сьогодні активно витісняється новим сімействомCortex.

- ARM9, ARM11 (з частотами до 1 ГГц) для більш потужних телефонів, кишенькових комп'ютерів і вбудованих рішень високої продуктивності.

- Cortex A – нове сімейство процесорів, що прийшло на зміну ARM9 і ARM11.

- Cortex M – нове сімейство процесорів, яке прийшло на зміну ARM7, також покликане зайняти нову для ARM нішу вбудованих рішень низької продуктивності. У сімействі присутні чотири значущі ядра: Cortex M0, Cortex M3, Cortex M4 і Cortex-M7.

У 2010 році виробник анонсував процесори Cortex-A15 під кодовою назвою Eagle. ARM стверджує, що ядро Cortex A15 на 40 відсотків продуктивніше на тій самій частоті, ніж ядро Cortex-A9 при однаковій кількості ядер на чіпі. Виріб, виготовлений по 28-нанометровому техпроцесу, має 4 ядра, може функціонувати на частоті до 2,5 ГГц і буде підтримуватися багатьма сучасними операційними системами.

Популярне сімейство мікропроцесорів xScale фірми Marvell (до 27 червня 2007 року – Intel) є розширенням архітектури ARM9, доповненої набором інструкцій Wireless MMX, спеціально розроблених фірмою Intel для підтримання мультимедійних додатків.

RISC (англ.restricted (reduced) instruction set computer – «комп'ютер зі скороченим набором команд») – архітектура процесора, у якому швидкодія збільшується за рахунок спрощення інструкцій (команд), щоб їх де-

кодування було простішим, а час виконання – меншим. Перші RISC-процесори навіть не мали інструкцій множення та ділення. Це дозволяє підвищити тактову частоту й робить більш ефективною суперскалярність (розпаралелювання інструкцій між декількома виконавчими блоками).

Набори інструкцій у більш ранніх архитектурах, для полегшення ручного написання програм на мовах ассемблерів або прямо в машинних кодах, а також для спрощення реалізації компіляторів, виконували якнайбільше роботи. Часто в ці набори включалися інструкції для прямого підтримання конструкцій мов високого рівня.

Інша особливість цих наборів: більшість інструкцій, як правило, допускали різні методи адресації (т. зв. «ортогональность системи команд (англ.) »). Наприклад, і операнди, і результат у арифметичних операціях доступні не тільки в регістрах, але й через безпосередню адресацію в пам'яті. Пізніше такі архітектури були названі CISC (англ. Complex instruction set computer).

Однак багато компіляторів не задіяли всі можливості таких наборів інструкцій, а на складні методи адресації вимагають більше часу через додаткові звернення до повільної пам'яті.

Було показано, що такі функції краще виконувати послідовністю більш простих інструкцій, при цьому архітектура процесора спрощується й у ньому залишається місце для більшої кількості регістрів, за рахунок яких можна скоротити кількість звернень до пам'яті.

У перших архітектурах, що зараховуються до RISC, більшість інструкцій для спрощення декодування мають однакову довжину й схожу структуру, арифметичні операції працюють тільки з регістрами, а робота з пам'яттю йде через окремі інструкції завантаження (load) і збереження (store). Ці властивості й дозволили краще збалансувати етапи конвеєризації, зробити конвеєри в RISC значно ефективнішими, підняти тактову частоту.

1.3 Характерні особливості RISC-процесорів

- Фіксована довжина машинних інструкцій (наприклад, 32 біта) і простий формат команди.

- Спеціалізовані команди для операцій з пам'яттю – читання або запису. Операції виду Read-Modify-Write («прочитати – змінити – записати») відсутні. Будь-які операції «змінити» виконуються тільки над вмістом регістрів (так звана архітектура load-and-store).

- Велика кількість регістрів загального призначення (32 і більше).

- Відсутність підтримання операцій виду «змінити» над укороченими типами даних. Так, наприклад, система команд DEC Alpha містила тільки операції над 64-бітовими словами й вимагала розроблення й подальшого виклику процедур для виконання операцій над байтами, 16- і 32-бітовими словами [2].

- Відсутність мікропрограм усередині самого процесора. Те, що в CISC-процесорі виконується мікропрограмами, у RISC-процесорі виконується як звичайний (хоча і поміщений у спеціальне сховище) машинний код, який не відрізняється принципово від коду ядра ОС і додатків. Так, наприклад, оброблення відмов сторінок у DEC Alpha й інтерпретація таблиць сторінок містилися в так званому PALcode (Privileged Architecture Library), вміщеному в постійному запам’ятовувальному пристрої (ПЗП). Заміною PALCode можна було перетворити процесор Alpha з 64- у 32-бітовий, а також змінити порядок байтів у слові й формат входів таблиць сторінок віртуальної пам'яті.

1.4 Кількість інструкцій

Дуже часто фраза «скорочений набір команд» розуміється як мінімізація кількості інструкцій у системі команд. Інструкцій у багатьох RISC-процесорів більше, ніж у CISC процесора. Деякі RISC-процесори на кшталт компю’терів фірми INMOS (англ.) мають набори команд не менше, ніж, наприклад, у CISC-процесорів IBM System / 370; і навпаки, CISC-процесор DECPDP-8 має тільки 8 основних і кілька розширених інструкцій.

Термін «скорочений» в назві описує той факт, що скорочений обсяг (і час) роботи, яка виконується кожною окремою інструкцією (як максимум один цикл доступу до пам'яті), тоді як складні інструкції CISC-процесорів можуть вимагати сотень циклів доступу до пам'яті для свого виконання.

Деякі архітектури, спеціально розроблені для мінімізації кількості інструкцій, дуже відрізняються від класичних RISC-архітектур і отримали інші назви: Minimal instruction set computer (MISC), Zero instruction set computer (ZISC), Ultimate RISC (так званий OISC), Transport triggered architecture (TTA) тощо.

1.5 Філософія RISC

У середині 1970-х різні дослідники (зокрема, з IBM) показали, що більшість комбінацій інструкцій і ортогональних методів адресації не використовувалися в більшості програм, що створювалися компіляторами того часу. Також було виявлено, що в деяких архітектурах з мікрокодовою реалізацією складні операції найчастіше були повільніше послідовності простіших операцій, що виконують ті самі дії. Це було викликано, зокрема, тим, що багато архітектур розроблялися в поспіху й добре оптимізувався мікрокод тільки тих інструкцій, які використовувалися частіше.

Оскільки багато реальних програм витрачають більшість свого часу на виконання простих операцій, багато розробників вирішили сфокусуватися на тому, щоб зробити ці операції максимально швидкими. Тактова частота процесора обмежена часом, який процесор витрачає на виконання найбільш повільної операції в процесі оброблення будь-якої інструкції; зменшення тривалості таких кроків дає загальне підвищення частоти, а також часто прискорює виконання інших інструкцій за рахунок більш ефективної конвеєризації.

Фокусування на простих інструкціях і веде до архітектури RISC, мета якої – зробити інструкції настільки простими, щоб вони легко конвеєризувалися й витрачали не більше одного такту на кожному кроці конвеєра на високих частотах.

Пізніше було відзначено, що найбільш значуща характеристика RISC у розділі інструкцій для оброблення даних і звернення до пам'яті полягає в тому, що звернення до пам'яті йде тільки через інструкції load і store, а всі інші інструкції обмежені внутрішніми регістрами. Це спростило архітектуру процесорів: дозволило інструкціям мати фіксовану довжину, спростило конвеєри й ізолювало логіку, яка має справу із затримками при доступі до пам'яті тільки у двох інструкціях. У результаті RISC-архітектури стали називати також архітектурою load/store.

РОЗДІЛ 2
ОПИС МІКРОКОНТРОЛЕРІВ СІМЕЙСТВА STM32

2.1 Призначення мікроконтролерів сімейства STM32 і короткий огляд їхніх характеристик

32-розрядні мікроконтролери випускаються багатьма виробниками, але найбільш широке поширення в наш час отримав продукт спільної франко-італійсько-японської фірми STMicroelectronics (STM).

Досить низька вартість, зручність програмування й наявність безкоштовного ПЗ сприяли його просуванню. Найбільш продуктивними в сімействі STM32 є мікроконтролери лінійки STM32F4 (рис. 2.1, табл.2.1).

Таблиця 2.1 – Опис мікроконтролерів STM32 і їх призначення

Серія	Тактова частота, МГц	Обчислювальна потужність, DMIPS*	Ядро	Опис
STM32F0	48	38	Cortex-M0	Бюджетний МК початкового рівня
STM32F1	24/36/48/72	61	Cortex-M3	МК загального призначення
STM32F2	120	150	Cortex-M3	Високопродуктивні МК
STM32F3	72	62	Cortex-M4	Сигнальний процесор DSP, FPU (операції з рухомою точкою)
STM32F4	84/168/180	210	Cortex-M4	Високопродуктивні МК, DSP, FPU
STM32L0	32	33	Cortex-M0 +	Низькі споживання й ціна
STM32L1	32	33	Cortex-M3	Наднизьке споживання енергії
STM32T	72	90	Cortex-M3	Контролер сенсорного екрану
STM32W	24	30	Cortex-M3	Для бездротового зв'язку (RF, ZigBee)

*DMIPS – мільйон операцій у секунду.

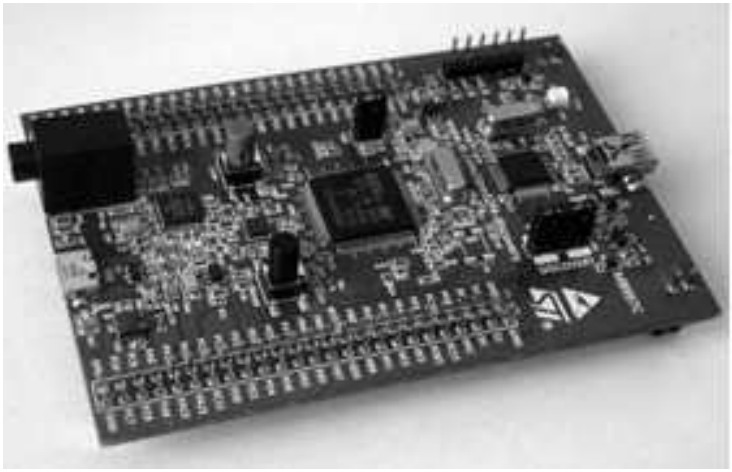


Рисунок 2.1 – Налагоджувальна плата STM32F4Discovery

Для навчання виробники пропонують так звані оцінні плати, на яких встановлюється той чи інший мікроконтролер зі схемами обв'язки й різною периферією. Обов'язковим атрибутом оцінних плат сьогодні є велика кількість контактів, які дозволяють отримати доступ практично до всіх портів мікроконтролера, а також інтерфейси у вигляді світлодіодів і кнопок. Також у багато оцінних плат вбудовується програматор, який дозволяє програмувати зовнішні мікросхеми.

У таблиці наведено оцінні плати початкового рівня лінійки STM32F4 (табл. 2.2).

Таблиця 2.2 – Оцінні плати початкового рівня лінійки STM32F4

Оцінна плата	Мікроконтролер	Периферійні пристрої
STM32F429I Discovery	STM32F429ZIT6 (180 МГц, 2 МБ Flash, 256 КБ ОЗУ, LQFP144)	Вбудований програматор / налаштовувач ST-LINK / V2, дисплей 2.4 "QVGA TFT, ОЗУ SDRAM 64 МБ, гіроскоп, USB-OTG, 6 світлодіодів, 2 кнопки
STM32F407 Discovery	STM32F407VGT6 (168 МГц, 1 МБ Flash, 192 КБ ОЗУ, LQFP100)	ST-LINK / V2, 3-осьовий акселерометр, цифровий мікрофон, USB-OTG, 24-розрядний аудіо-ЦАП з підсилювачем класу D, 8 світлодіодів, 2 кнопки
STM32F401C Discovery	STM32F401CVT6 (84 МГц, 256 КБ Flash, 64 КБ ОЗУ, LQFP100, низьке енергоспоживання)	ST-LINK / V2, гіроскоп, компас, цифровий мікрофон, USB-OTG, 24-розрядний аудіо-ЦАП з підсилювачем класу D, 8 світлодіодів, 2 кнопки
STM32F401 Nucleo	STM32F401RET6 (84 МГц, 512 КБ Flash, 96 КБ ОЗУ, LQFP64, низьке енергоспоживання)	ST-LINK / V2, рознім для підключення Shield Arduino, 2 світлодіоди, 2 кнопки

2.2 Знайомство з налагоджувальною платою STM32F4Discovery

Для ознайомлення з 32-розрядними мікроконтролерами буде використовуватися налагоджувальна плата STM32F407Discovery (рис. 2.2).

Ця плата має дві модифікації: MB997C і MB997B. Перша прийшла на зміну другій та має на борту більш сучасну мікросхему акселерометра LIS3DSH замість застарілої LIS302DL [3].

Рисунок 2.2 – Налагоджувальна плата STM32F407

Компоненти оцінної плати STM32F4Discovery та їх призначення наведено на рис. 2.3.

Рисунок 2.3 – Призначення компонентів STM32F4Discovery

На звороті плати знаходяться штирові виводи (рис. 2.4).

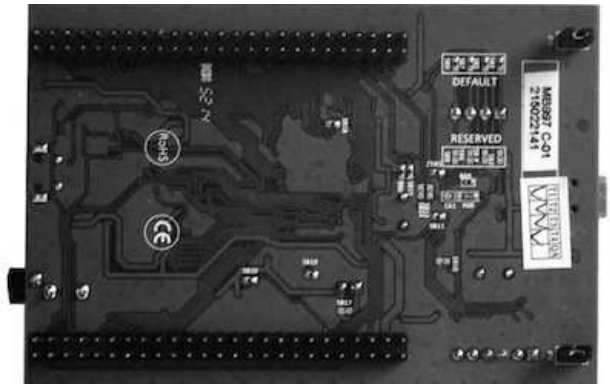


Рисунок 2.4 – Зворотний бік оцінної плати

2.3 Характеристики й можливості налагоджувальної плати STM32F4Discovery

Основні характеристики плати:

- 32-бітний мікроконтролер STM32F407VGT6 з ядром ARM Cortex-M4F з 1 МБ пам'яті програм і 193 КБ ОЗУ в 100-виводному корпусі LQFP100 з тактовою частотою 168 МГц. Вбудовані операції з рухомою точкою (FPU);
- вбудований програматор / налаштовувач ST-LINK / V2 з можливістю вибору режиму роботи (дозволяє програмувати зовнішні мікросхеми, використовуючи SWD-коннектор для програмування й налаштування);
- живлення плати: через шину USB або від зовнішнього джерела живлення5 В;
- живлення для зовнішніх пристроїв: 3 В і 5 В;
- 3-осьовий MEMS-акселерометр на базі мікросхеми LIS302DL компанії ST;
- всенаправлений цифровий MEMS-мікрофон на базі мікросхеми MP45DT02 компанії ST;
- аудіо ЦАП CS43L22 із вбудованим підсилювачем класу D;
- вісім світлодіодів: LD1 (червоний / зелений) для індикації активності шини USB, LD2 (червоний) для живлення 3,3 В; 4 користувацькі діоди: LD3 (помаранчевий), LD4 (зелений), LD5 (червоний) і LD6 (синій); 2 діоди USB OTG: LD7 (зелений) для VBus і LD8 (червоний) при перевантаженні;
- дві кнопки (Reset і User);
- USB OTG (On The Go) з рознімом micro-AB;
- вивідні колодки для всіх контактів введення-виведення мікроконтролера для швидкого підключення до макетної плати та простого проведення вимірювань.

Великим плюсом є наявність у мікроконтролері модуля для роботи з числами з рухомою точкою, що збільшує швидкість оброблення в додатках, пов'язаних, наприклад, зі спектральним аналізом (для обчислення ШПФ (швидке перетворення Фур'є)) або ж у БПЛА (швидке перетворення Лапласа) для алгоритмів орієнтації в просторі. До дрібних недоліків можна віднести лише відсутність розніму JTAG для тестування, використовування зовнішнього програматора / налаштовувача ST-LINK.

Спрощену структурну схему налагоджувальної плати наведено на рис. 2.5.

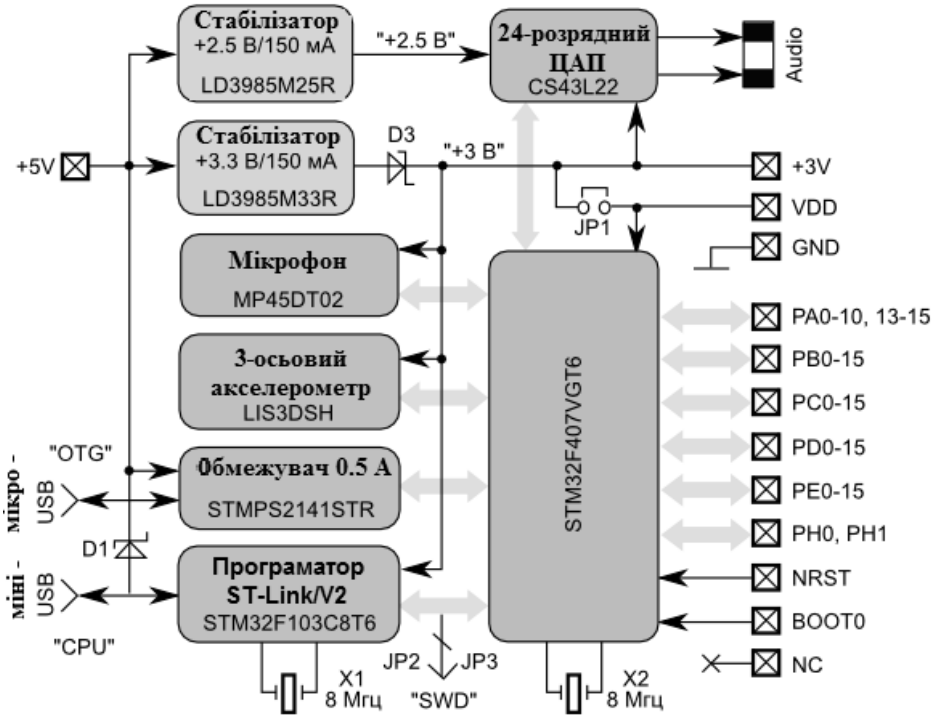


Рисунок 2.5 – Структурна схема оцінної плати STM32F4Discovery

З рис. 2.5 видно, що через діод Шотткі D3 напруга +3 В не є строго стабілізованою. Струм навантаження через контакт «+ 3 В» не повинен перевищувати 150 мА. Якщо замість джампера J1 підключити амперметр, то можна побачити, що споживаний мікроконтролером ток 80…130 мА.

Живлення 5 В подається через міні-USB рознім з комп'ютера. Через діод D1 на мікро-USB-рознімі «OTG» буде трохи менша напруга.

На колодці штирових контактів є контакт «+5 В», до якого можна підключити зовнішнє джерело живлення. Завдяки діоду D1 є можливість одночасно використовувати внутрішній і зовнішній джерела живлення. Програматор ST-Link реалізований на мікроконтролері STM32F103C8T6. Основний програмований мікроконтролер підключається для програмування через два джампери J2 і J3. Для програмування зовнішніх мікросхем, використовуючи рознім «SWD», ці джампери необхідно видалити. Для роботи цифрового MEMS-мікрофона, 3-осьового акселерометра й аудіо-ЦАП задіяно кілька ліній портів мікроконтролера. Дані передаються по шинам SPI, I2C, I2S. У таблиці 2.3 наведено задіяні й вільні порти.

Таблиця 2.3 – Опис портів мікроконтролера

Вивід порту	Функції порту
PA0	Кнопка «User»
BOOT0	Вхід бутлоудера, сигнал BOOT0
PB2	Вхід бутлоудера, сигнал BOOT1
PA1-PA3, PA8, PA15, PB0, PB1, PB4, PB5, PB7, PB8, PB11, PB13-PB15, PC1, PC2, PC4-PC6, PC8, PC9, PC11, PD0-PD3, PD6-PD11, PE2, PE4-PE15	Вільні лінії I / O, толерантні до 5 В, максимальне навантаження ± 25 мА, pull-up / down резистори 30...50 кОм (всього 46 ліній)
PB12	Вільна лінія з pull-up/down резистором 8...15 кОм
PC13	Вільна лінія з навантаженням ± 3 мА
PA3-PA6, PB6, PB9, PC7, PC10, PC12, PD4	Стереофонічний аудіо-ЦАП CS43L22
PA5-PA7, PE0, PE1, PE3	3-осьовий акселерометр LIS3DSH
PA9-PA12, PC0, PD5	Рознім мікро-USB (OTG)
PA13, PA14, PB3	Рознім програматора SWD
PB10, PC3	Вбудований цифровий мікрофон MP45DT02
PC14, PC15	Кварцовий резонатор 32 кГц (є місце)
PD12-PD15	Зелений, оранжевий, червоний, синій світлодіоди
PH0, PH1	Кварцовий резонатор 8 МГц для МК
NRST	Зовнішнє початкове скидання МК
+ 3 V, + 5 V, VDD, GND, NC	Ланцюги живлення 3 В, 5 В, МК, «земля», порожній контакт

Щоб перевірити працездатність налагоджувальної плати, потрібно:

- Упевнитися, що на платі встановлено перемички JP1 і CN3.

- Плату STM32F407Discovery потрібно підключити до комп'ютера, використовуючи USB-кабель типу A / mini-B через міні-USB-рознім CN1 програматора ST-Link на платі для подавання живлення. Засвітиться червоний світлодіод LD2 (PWR) і почнуть мигати чотири світлодіоди (зелений, помаранчевий, червоний, синій), що знаходяться між кнопками B1 і B2.

- Натиснути призначену для користувача клавішу B1, яка включає MEMS-акселерометр. Чотири кольорові світлодіоди показуватимуть напрямок руху плати й швидкість. При підключенні до комп'ютера через другий USB-рознім на платі CN5 з використанням кабелю типу A / micro-B плата розпізнається як стандартний маніпулятор «миша».

- Програмне забезпечення для програмування мікроконтролера, різна документація на налагоджувальну плату STM32F407Discovery.

- Демонстраційні приклади, що дозволяють ознайомитися з особливостями сімейства мікроконтролерів STM32F4, доступні на офіційному сайті STMicroelectronics.

РОЗДІЛ 3
ОСНОВИ МОВИ ПРОГРАМУВАННЯ С

3.1 Знайомство з мовою програмування С. Область застосування та основні можливості

Мова С – це універсальна мова програмування, для якої характерні економічність виразів, сучасний набір операторів і типів даних. Мова С не є ні мовою «дуже високого рівня», ні «великою» мовою і не призначається для деякої спеціальної області застосування, але відсутність обмежень робить її для багатьох завдань зручною й ефективною. Мова С не пов'язана з будь-якими певними апаратними засобами або системами, на ній легко писати програми, які можна пропускати без змін на будь-який ЕОМ, що має С-компілятор. Мова С є універсальною мовою програмування. Вона спочатку з'явилася в операційній системі UNIX і розвивалася як основна мова систем, сумісних із ОС UNIX.

Проте сама мова не пов'язана з якоюсь однією операційною системою або машиною, і хоча її називають мовою системного програмування, тому що вона зручна для написання операційних систем, вона може використовуватися для написання будь-яких великих обчислювальних програм, програм для оброблення текстів і баз даних. Мова С – це мова відносно «низького рівня» [4].

Це означає, що С має справу з об'єктами того самого виду, що й більшість ЕОМ, а саме з символами, числами й адресами. Вони можуть об'єднуватися й пересилатися за допомогою звичайних арифметичних і логічних операцій, здійснюваних реальними ЕОМ.

У мові С відсутні операції, що мають справу безпосередньо зі складовими об'єктами, такими як рядки символів, множини, списки, або з масивами, які розглядаються як ціле. Тут, наприклад, немає жодного аналога операцій PL / 1, які оперують масивами й рядками. Мова не надає жодних інших можливостей розподілу пам'яті, окрім статичного визначення й механізму стеків, що забезпечується локальними змінними функцій. Сама по собі мова С не забезпечує жодних можливостей введення-виведення. Усі ці механізми високого рівня повинні забезпечуватися явним викликом функцій.

Мова С пропонує тільки прості послідовні конструкції керування: перевірки, цикли, групування й підпрограми, але не мультипрограмування, паралельні операції, синхронізацію або співпрограми. Утримання мови в скромних розмірах дає реальні переваги. Оскільки мова С відносно невелика, вона не вимагає багато місця для свого опису й може бути швидко вивчена.

3.2 Основи мови програмування С

Ключові слова

Зарезервовані для використання в якості ключових слів і не можуть використовуватися іншим чином:

int	extern	else
char	register	for
float	typedef	do
double	static	while
struct	goto	switch
union	return	case
long	sizeof	default
short	break	entry
unsigned	continue	
auto	if	

Ключове слово entry в наш час не використовується будь-яким компілятором, воно зарезервовано для використання в майбутньому. У деяких реалізаціях резервуються також слова fortran і asm.

Типи даних у С

У мові С передбачено кілька основних типів об'єктів.

Символьний

Об'єкти, описані як символи (char), досить великі, щоб зберігати будь-який член з відповідного цій реалізації внутрішнього набору символів, і якщо дійсний символ з цього набору символів зберігається в символьній змінній, то її значення еквівалентно цілому коду цього символу. У символьних змінних можна зберігати й інші величини, але реалізація буде машинно-залежною.

Приклад оголошення змінної:

```
char symbol;
symbol = 'f';
```

Цілий

Можна використовувати до трьох розмірів цілих, описуваних як short int, int і long int, uint8_t, uint16_t, uint32_t. Довгі цілі використовують не менше пам'яті, ніж короткі, але в конкретній реалізації може виявитися, що короткі цілі, або довгі цілі, або ті та інші будуть еквівалентні простим цілим. «Прості» цілі мають природний розмір, що передбачається архітектурою використовуваної машини; інші розміри вводяться для задоволення спеціальних потреб.

Приклад оголошення змінної і її використання:

```
int n;
n = 1000;
```

Беззнаковий

Цілі без знака, що описуються як unsigned, підпадають під дію законів арифметики за модулем 2n, де n – число бітів у їх поданні.

Речовий

Дійсні одинарної точності (float) і подвійної точності (double) в деяких реалізаціях можуть бути синонімами (float займає 32 біта пам'яті, а double – 64). У мові немає логічного типу даних, а в якості логічних значень використовуються цілі «0» – «не істина» і «1» – «істина» (при перевірках будь-яке ціле, яке не дорівнює 0, трактується як «істина»). Оскільки об'єкти згаданих вище типів можуть бути розумно інтерпретовані як числа, ці типи називаються арифметичними. Типи char і int всіх розмірів спільно називатимуться цілочисловими. Типи float і double спільно називатимуться речовими типами.

Крім основних арифметичних типів, існує концептуально нескінченний клас похідних типів, які утворюються з основних типів, так:

- масиви об'єктів більшості типів;
- функції, які повертають об'єкти заданого типу;
- покажчики на об'єкти даного типу;
- структури, що містять послідовність об'єктів різних типів;
- структури, здатні утримувати один з декількох об'єктів різних типів.

Процедури й функції

Часто на практиці ефективніше звести деякі повторювані фрагменти коду в окремі блоки (процедури й функції). Процедури й функції можна повторно використовувати в програмних кодах.

Процедури оголошуються за допомогою ключового слова void.

Приклад процедури, яка реалізує затриманя запалювання світло діодів, має вигляд:

```
// Затримання
void Delay (uint32_t nCount)
{
  while (nCount--)
{
}
}
```

Delay – ім'я процедури (можна використовувати будь-яке), nCount – вхідна змінна величини затримання в часі.

Цю процедуру можна дуже просто викликати з основної програми в такий спосіб:

```
Delay (1000);
```

Оголошення та використання функцій на прикладі:

```
int F (int x)
{
int y;
y = x * x;
return y;
}
```

Спочатку потрібно оголосити тип функції (у цьому випадку int). Далі оголошується ім'я функції (тут – F, можна будь-яке інше, крім зарезервованих слів), x – вхідна змінна функції. За допомогою зарезервованого слова return привласнюється значення внутрішньої змінної y функції F.

3.3 Оператори мови С

За винятком особливих випадків, оператори виконуються послідовно.

Операторний вираз

Більшість операторів є операторними виразами, які мають форму виразу.

Зазвичай операторні вирази є привласнюванням або зверненнями до функцій.

Приклад:

```
a = 0;
b = 10;
z = a + b;
```

Складений оператор (або блок)

З тим, щоб допустити можливість використання декількох операторів там, де очікується наявність тільки одного, передбачається складений оператор (який також називають «блоком»).

Складений оператор:

```
{Список_описів список_операторів}
список_описів:
                опис
                опис список_описів
список_операторів:
                оператор
                оператор список_операторів
```

Якщо який-небудь ідентифікатор зі списку описів був описаний раніше, то під час виконання блоку зовнішній опис заглушується і знову стає чинним після виходу з блоку. Будь-яка ініціалізація автоматичних і реєстрових змінних проводиться при кожному вході в блок через його початок.

3.4 Умовні оператори

Є дві форми умовних операторів:

```
if (вираз) оператор
if (вираз) оператор else оператор
```

У першому випадку перевіряється вираз, і якщо він відповідає умові, то виконується оператор. У другому випадку, якщо вираз відповідає умові, виконується перший оператор, якщо ні – другий.

Як завжди, двозначність else вирішується зв'язуванням else з останнім зустрічним if, у якого немає else.

Приклади використання:

```
// Ініціалізація змінних
int a = 10;
int b = 10;
int c = 1;
int z;
// Розрахунок z згідно з умовою
if (a> 0) z = a + b;
З конструкцією else
if (a> 0) z = a + b else z = ba;
```

Якщо в тілі умови потрібно виконати кілька операторів, то оператори беруться в операторні дужки:

```
if (a> 0)
{
 a = -20;
 b = -10
 z = a + b;
}
else
{
 b = 0;
 a = 8;
 z = b*a;
```

Умовний-вираз:
Логічний-або-вираз
логічний-АБО-вираз ? вираз: умовний-вираз

Обчислюється перший вираз. Якщо він не дорівнює 0, то результатом є значення другого виразу, в іншому випадку – значення третього виразу. Обчислюється тільки один з двох останніх операндів: другий або третій. Якщо другий і третій операнди арифметичні, то виконуються звичайні арифметичні перетворення, що ведуть до деякого загального типу, який і буде типом результату. Якщо обидва операнди мають тип void або є структурами чи об'єднаннями одного й того самого типу, або представляють собою покажчики на об'єкти одного й того самого типу, то результат буде мати той самий тип, що й операнди. Якщо один із операндів має тип «покажчик», а інший є константою 0, то 0 зводиться до типу «покажчик», цей самий тип буде мати й результат. Якщо один операнд є покажчиком на void, а другий – покажчиком іншого типу, то останній перетвориться на покажчик на void, який і буде типом результату [5].

Приклад:

```
m = (a> 0)? b: -b;
```

3.5 Цикли в С. Цикл while

Оператор while

Оператор while має форму «while (вираз) оператор;». Оператор виконується повторно доти, поки значення виразу задовольняє умові. Перевірка проводиться перед кожним виконанням оператора.

Приклад:

```
while (i <100) i = i + 1;
```

Оператор do

Оператор do має форму «do оператор while (вираз);».

Оператор виконується повторно доти, поки значення виразу не стане дорівнювати необхідній умові. Перевірка проводиться після кожного виконання оператора.

3.6 Цикл for

Оператор for

Оператор for має форму «for (вираз 1; вираз 2; вираз 3) оператор;».

Таким чином, перший вираз визначає ініціалізацію циклу; другий специфікує перевірку, виконувану перед кожною ітерацією, так що вихід з циклу відбувається тоді, коли значення виразу стає нулем; третій вираз часто задає збільшення параметра, який обчислюється після кожної ітерації.

Будь-яке вираження або всі вони можуть бути опущені. Якщо відсутній другий вираз, то пропозиція з while вважається еквівалентним while (1); інші відсутні вирази просто опускаються з наведеного вище розширення.

Приклад:

```
int s = 0;
int i;
for (i = 1; i <10; i ++)
s = s + 1;
```

3.7 Оператор switch

Оператор switch (перемикач) викликає передавання керування до одного з декількох операторів залежно від значення виразу. Оператор має форму «switch (вираз) оператор;».

У виразі проводяться звичайні арифметичні перетворення, результат повинен мати тип int. Оператор зазвичай є складеним. Будь-який оператор всередині цього оператора може бути позначений одно- або кількаваріантним префіксом case, що має форму «case константний вираз», де константний вираз має мати тип int. Жодні дві варіантні константи в одному і тому самому перемикачі не можуть мати однакове значення. Точне визначення константного виразу наводиться нижче. Крім того, може бути присутнім один операційний префікс виду «default:».

При виконанні оператора switch обчислюється вираз, що входить у нього й порівнюється з кожною варіантною константою. Якщо одна з варіантних констант виявляється такою, що дорівнює значенню цього виразу, то керування передається оператору, який йде за збіжним варіантним префіксом. Якщо жодна з варіантних констант не збігається зі значенням виразу й при цьому наявний префікс default, то керування передається оператору, позначеному цим префіксом. Якщо жоден з варіантів не підходить і префікс default відсутній, то жоден з операторів у перемикачі не виконується.

Самі по собі префікси case і default не змінюють виконання програми, програма виконується послідовно, поки не зустрінеться явне переда-

вання керування. Для виходу з перемикача є оператор break. Зазвичай оператор, який входить у перемикач, є складеним. Описи можуть з'являтися на початку цього оператора, але ініціалізації автоматичних і реєстрових змінних будуть неефективними.

Приклад:

```
switch (regim) {
case 'x': regx ++;
case 'X': case 'Y': regY ++; break;
case '-': regx = 0; break;
default: err ("Помилка"); goto next;
}
```

3.8 Оператори break, continue, return

3.8.1 Оператор break

Оператор break; викликає завершення виконання найменшого оператора, що охоплює цей оператор, while, do, for або switch; керування передається оператору, що йде за завершеним оператором.

3.8.2 Оператор continue

Оператор continue викликає до передавання керування на частину, яка продовжує цикл найменшого оператора, що охоплює цей оператор, while, do або for, тобто на кінець циклу.

```
while (...) {| do {| for (...) {
... | ... | ...
contin:; | contin:; | contin:;
} | } While (...); | }
```

3.8.3 Оператор повернення

Повернення з функції до викликальної програми здійснюється за допомогою оператора return, який має одну з таких форм:

```
return;
return вираз;
```

У першому випадку значення, яке повертається, не визначено. У другому – до викликальної функції повертається значення виразу. Якщо потрібно, вираз перетвориться на тип функції, у якій він з'являється, як у разі привласнення. Потрапляння на кінець функції еквівалентно поверненню без значення. Повертати можна значення арифметичного типу, а також структуру (але не масив).

3.9 Префіксні оператори інкремента й декремента

Унарний вираз, перед яким стоїть «++» або «--», є унарним виразом. Операнд збільшується (++) або зменшується (--) на 1. Значенням виразу є значення його операнда після збільшення (зменшення).

3.10 Бітові й логічні оператори

Операції можна класифікувати за кількістю операндів: унарні – вплив на один операнд, бінарні – впливають на два операнди, тернарні – вплив на три операнди.

3.10.1 Оператор побітового заперечення

Операнд оператора «~» повинен мати цілочисловий тип, результат – доповнення операнда до одиниць за всіма розрядами. Виконується цілочислове підвищення типу операнда. Якщо операнд беззнаковий, то результат – вирахування його значення з найбільшого числа підвищеного типу. Якщо операнд знаковий, то результат обчислюється за допомогою зведення «підвищеного операнда» до беззнакову типу, виконання операції ~ і зворотного зведення його до знакового типу. Тип результату – підвищений тип операнда.

3.10.2 Оператор логічного заперечення

Операнд оператора «!» повинен мати арифметичний тип або бути вказівником. Результат дорівнює 1, якщо порівняння операнда з 0 дає істину, і дорівнює 0 у протилежному випадку. Тип результату – int.

3.10.3 Мультиплікативні оператори

Мультиплікативні оператори «*», «/» і «%» виконуються зліва направо.

Мультиплікативний-вираз:
вираз, зведений до типу
мультиплікативний-вираз * вираз-зведений-до-типу
мультиплікативний-вираз / вираз-зведений-до-типу
мультиплікативний-вираз% вираз-зведений-до-типу

Операнди операторів «*» та «/» повинні бути арифметичного типу, оператора «%» – цілочислового типу. Над операндами здійснюються звичайні арифметичні перетворення, які зводять їх значення до типу результату.

Бінарний оператор «*» позначає множення. Бінарний оператор «/» отримує частку, а «%» – залишок від ділення першого операнда на другий; якщо другий операнд є 0, то результат не визначений. В іншому випадку завжди виконується співвідношення (a / b) * b + a% b = a. Якщо обидва операнди не негативні, то залишок негативний і менший від дільника; в іншому випадку стандарт гарантує тільки одне: абсолютне значення залишку менше від абсолютного значення дільника.

3.10.4 Адитивні оператори

Адитивні оператори «+» і «-« виконуються зліва направо. Якщо операнди мають арифметичний тип, то здійснюються звичайні арифметичні перетворення. Для кожного оператора існує ще кілька додаткових сполучень типів.

Адитивний-вираз:
мультиплікативний-вираз
адитивний-вираз + мультиплікативний-вираз
адитивний-вираз - мультиплікативний-вираз

Результат виконання оператора «+» є сума його операндів.

3.10.5 Оператори зсуву

Оператори зсуву «і» виконуються зліва направо. Для обох операторів кожен операнд повинен мати цілочисловий тип, і кожен з них піддається цілочисловому підвищенню. Тип результату збігається з підвищеним ти-

пом лівого операнда. Результат не визначений, якщо правий операнд негативний або його значення перевищує число бітів у типі лівого виразу або дорівнює йому.

Вираз зсуву:
адитивний-вираз
вираз зсуву >> адитивний-вираз
вираз зсуву << адитивний-вираз

Значення E1 << E2 дорівнює значенню E1 (оскільки воно розглядалося як ланцюжок бітів), зсунутому ліворуч на E2 бітів; за відсутності переповнення така операція еквівалентна множенню на 2E2. Значення E >> E2 дорівнює значенню E1, зсунутому праворуч на E2 бітові позиції. Якщо E1 беззнакове або має позитивне значення, то праве зсунення еквівалентне діленню на 2E2, в іншому випадку результат залежить від реалізації.

Приклад використання зсуву вправо:

```
int i = 255; // У двійковій системі i = 11111111
int c; // Змінна для зберігання результату зсуву
c = i >> 4; // Зрушуємо біти числа i на 4 розряди праворуч
```

У результаті c = 15 (у двійковій системі c = 1111). За допомогою інженерного калькулятора нескладно перевести будь-яке число в різні системи числення (десяткова, шістнадцяткова, двійкова, вісімкова).

Приклад використання зсуву вліво:

```
int i = 255; // У двійковій системі i = 11111111
int c; // Змінна для зберігання результату зсуву
c = i << 4; // Зрушуємо біти числа i на 4 розряди ліворуч
```

У результаті c = 4080 (у двійковій системі c = 111 111 110 000).

3.10.6 Оператори відношення

Оператори відношень виконуються зліва направо, проте ця властивість ледве може виявитися корисною; згідно з граматикою мови вираз a < b < c трактується так само, як (a < b) < c, а результат обчислення a < b

Може бути тільки 0 або 1.
вираз-відношення:
вираз-зсуву
вираз-відношення <вираз-зсуву вираз-відношення >
вираз-зсуву вираз-відношення <= вираз-зсуву
вираз-відношення > = вираз-зсуву

Оператори «<» (менше), «>» (більше), «<=» (менше або дорівнює) і «>=» (більше або дорівнює) видають 0, якщо специфіковане відношення помилкове, та 1, якщо воно істинне. Тип результату – int. Над арифметичними операндами виконуються звичайні арифметичні перетворення.

Можна порівнювати покажчики на об'єкти одного й того самого типу (без урахування кваліфікаторів); результат буде залежати від відносного розташування в пам'яті. Однак допускається порівняння покажчиків на різні частини одного й того самого об'єкта: якщо два покажчики вказують на один і той самий простий об'єкт, то вони рівні; якщо вони вказують на елементи однієї структури, то покажчик на елемент з більш пізнім оголошенням у структурі більший; якщо покажчики вказують на елементи одного й того самого об'єднання, то вони рівні; якщо покажчики вказують на елементи деякого масиву, то порівняння цих покажчиків еквівалентно порівнянню їхніх індексів. Якщо P вказує на останній елемент масиву, то P + 1 більше, ніж P, хоча P + 1 вказує за межі масиву. В інших випадках результат порівняння не визначений. Ці правила дещо послабили обмеження, встановлені в першій редакції мови.

Вони дозволяють порівнювати покажчики на різні елементи структури та легалізують порівняння з покажчиком на місце, яке розташоване безпосередньо за кінцем масиву.

3.10.7 Оператори рівності

Вираз-рівності:
вираз-відношення
вираз-рівності == вираз-відношення
вираз-рівності! = вираз-відношення

Оператори «==» (дорівнює) і «! =» (не дорівнює) аналогічні операторам відношення з тією лише різницею, що мають більш низький пріоритет. (Таким чином, a < b == c < d є 1 тоді й тільки тоді, коли відношення a < b і c < d або обидва істинні, або обидва хибні).

Оператори рівності підкоряються тим самим правилам, що й оператори відношення. Крім того, вони дають можливість порівнювати покажчик з цілочисловим константним виразом, значення якого дорівнює нулю, і з покажчиком на void.

3.10.8 Оператор побітового І

І-вираз:
вираз-рівності
І-вираз & вираз-рівності

Виконуються звичайні арифметичні перетворення; результат – побітове І операндів. Оператор застосовується тільки до цілочислових операндів.

3.10.9 Оператор побітового АБО

Виключальний-АБО-вираз:
І-вираз
виключальний-АБО-вираз ^ І-вираз

Виконуються звичайні арифметичні перетворення; результат – побітове виключальне АБО операндів. Оператор застосовується тільки до цілочислових операндів.

3.10.10 Оператор побітового АБО

АБО-вираз:
виключальний-АБО-вираз
АБО-вираз | виключальний-АБО-вираз

Виконуються звичайні арифметичні перетворення; результат – побітове АБО операндів. Оператор застосовується тільки до цілочислових операндів.

3.10.11 Оператор логічного І

Логічний-І-вираз:
АБО-вираз
логічний-І-вираз && АБО-вираз

Оператори «&&» виконуються зліва направо. Оператор «&&» видає 1, якщо обидва операнди не дорівнюють нулю, і 0 в іншому випадку. На відміну від «&», «&&» гарантує, що обчислення проводитимуться зліва

направо: обчислюється перший операнд з усіма побічними ефектами; якщо він дорівнює 0, то значення виразу є 0. В іншому випадку обчислюється правий операнд, і якщо він дорівнює 0, то значення виразу є 0, в іншому випадку воно дорівнює 1. Операнди можуть належати до різних типів, але при цьому кожен з них повинен мати або арифметичний тип, або бути покажчиком. Тип результату – int.

3.10.12 Оператор логічного АБО

Логічний-АБО-вираз:
логічний-І-вираз
логічний-АБО-вираз || логічний-І-вираз

Оператори «||» виконуються зліва направо. оператор «||» видає 1, якщо принаймні один з операндів не дорівнює нулю, і 0 в іншому випадку. На відміну від «|», оператор «||» гарантує, що обчислення проводитимуться зліва направо: обчислюється перший операнд, включаючи всі побічні ефекти; якщо він не дорівнює 0, то значення виразу є 1. В іншому випадку обчислюється правий операнд, і якщо він не дорівнює 0, то значення виразу є 1, в іншому випадку воно дорівнює 0. Операнди можуть належати різним типам, але операнд повинен мати або арифметичний тип, або бути покажчиком. Тип результату – int.

3.10.13 Вирази привласнювання

Існує кілька операторів привласнювання; вони виконуються справа наліво.

Вираз-привласнювання:
умовний-вираз
унарний-вираз оператор-привласнювання вираз-привласнювання
оператор-привласнювання: один з
/ =% = + = - = «=» = & = ^ =: =

Оператори привласнювання в якості лівого операнда вимагають lvalue, причому такий, що модифікується; це означає, що він не може бути масивом або мати незавершений тип, або бути функцією. Тип лівого операнда, крім того, не може мати кваліфікатора const; і, якщо він є структурою або сукупністю, у них не повинно бути елементів або піделементів (для вкладених структур або сукупностей) з кваліфікаторів const.

Тип виразу привласнювання відповідає типу його лівого операнда, а значення дорівнює значенню його лівого операнда після завершення

привласнювання. У простому привласнюванні з оператором «=» значення виразу заміщає об'єкт, до якого звертається lvalue. При цьому повинно виконуватися одна з таких умов: обидва операнди мають арифметичний тип (якщо типи операндів різні, правий операнд зводиться до типу лівого операнда); обидва операнди є структурами або сукупностями одного й того самого типу; один операнд є покажчиком, а інший – покажчиком на void; лівий операнд – покажчик, а правий – константний вираз із значенням 0; обидва операнди – покажчики на функції або об'єкти, що мають однаковий тип (за винятком можливої відсутності const або volatile у правого операнда).

Вираз E1 op = E2 еквівалентний виразу E1 = E1 op (E2) з одним винятком: E1 обчислюється тільки один раз.

3.11 Масиви в С

Масиви даних на мові С визначаються так:

Тип масиву D [i], де D – ім'я масиву; тип масиву – int, float, char тощо; i – число елементів масиву.

Масив можна конструювати з об'єктів арифметичного типу, покажчиків, структур і сукупностей, а також інших масивів (генеруючи при цьому багатовимірні масиви). Будь-який тип, з якого конструюється масив, повинен бути завершеним, він не може бути, наприклад, структурою або масивом незавершеного типу. Це означає, що для багатовимірного масиву порожньою може бути тільки перша розмірність. Незавершений тип масиву отримує своє завершення або в іншому оголошенні цього масиву, або при його ініціалізації. Наприклад, запис «float fa [17], * afp [17];» оголошує масив з чисел типу float і масив з покажчиків на числа типу float.

Приклад:

```
int a [16] = {1, 3, 4, 5, 6, 7, 8, 9, 0, 5, 3, 4, 5, 7, 6, 4}
```

Звертатися до елементів масиву можна за їх індексами, наприклад a [1] дорівнюватиме 1, a [16] дорівнюватиме 4. Максимальне число елементів – 16.

РОЗДІЛ 4
ПОРТИ ВВЕДЕННЯ-ВИВЕДЕННЯ STM32F4DISCOVERY. ЇХ ПРИЗНАЧЕННЯ І НАЛАШТУВАННЯ

Порти введення-виведення – найважливіша частина будь-якого мікроконтролера, без них неможливий жоден інтерфейс з мікроконтролером (отримання інформації від кнопок, датчиків, приймання й передавання даних, підключення LCD-дисплея тощо) [6].

Скільки не було б у мікроконтролера пам'яті, периферії, якою б високою не була тактова частота – все це не має значення, якщо він не може взаємодіяти із «зовнішнім світом». А взаємодія здійснюється через ці самі порти введення-виведення. Далі для стислості вони будуть називатися просто портами. На мікроконтролері STM32F4Discovery є порти A, B, C, D, E, F, G, H, I. Кожен порт містить у собі певну кількість виводів (див. документацію). Будь-який з виводів можна використовувати в різних режимах. Можна підключати до них різні датчики, кнопки, світлодіоди, USB-UART тощо.

Існує два основні режими роботи виводів мікроконтролера: вхід і вихід. Коли вивід (пін) контролера налаштований на вихід, до нього можна підключити будь-який споживач: світлодіод, пискавку тощо. Потрібно розуміти, що виводи мікроконтролера не розраховані на велике навантаження. Максимальний струм, який може пропустити через себе один пін, складає ~ 20 мА. Якщо потрібно підключати щось із більш високим енергоспоживанням, то потрібно робити це через транзисторний ключ. В іншому випадку вивод порту (або весь порт) вийде з ладу і перестане виконувати свої функції.

Щоб убезпечити вивід порту при підключенні світлодіода або кнопки, можна підключити їх через резистор номіналом приблизно 220 Ом. Таким чином, при напрузі живлення 3,3 В навіть при короткому замиканні виводу на землю струм не перевищить критичного значення.

Другий режим роботи виводу контролера – це вхід. Завдяки цьому режиму можна зчитувати, наприклад, стан кнопок, перевіряючи, є на виводі напруга або немає. На налагоджувальній платі STM32F4Discovery вже підключені призначені для користувача світлодіоди та кнопка [7].

Для роботи з будь-яким портом мікроконтролера його потрібно насамперед увімкнути (подати тактові імпульси мікроконтролера, іноді кажуть «затактувати»). Справа в тому, що в мікроконтролері STM32F4 за умовчанням вимкнені всі порти з метою зниження енергоспоживання.

За умовчанням усі порти налаштовані на введення. При використанні виводів портів потрібно їх відповідним чином налаштувати. Під час налаштування виводу порту на виведення потрібно налаштувати частоту порту (наприклад, щоб мигнути світлодіодом). Частота порту, іншими словами швидкість комутації порту, – це кількість перемикань вивід порту мікроконтролера у секунду (за умовчанням мінімальна частота 2 МГц, максимальна – 100 МГц).

Також під час налаштовування порту можна вибрати різні режими: звичайний або альтернативний Push-pull і звичайний або альтернативний з відкритим колектором Open drain.

У звичайному режимі можна розпоряджатися виводом на свій розсуд, наприклад встановити одиницю або нуль за допомогою свого коду.

В альтернативному режимі цей пін мікроконтролера передається в розпорядження будь-якої периферії контролера, наприклад UART, SPI, I2C, та інших пристроїв, що потребують виводів.

У режимі push-pull вивід завжди знаходиться в одному з двох станів: на ньому завжди або земля. або повна напруга живлення. У режимі відкритого колектора Open drain вивід просто ні до чого не підключений усередині мікроконтролера.

Аналоговий режим призначений для роботи АЦП. Якщо потрібно, щоб АЦП міг вимірювати, використовуючи цей вивід, потрібно вибрати цей режим.

Існує також режим налаштовування з підтягненням виведення до нуля або одиниці, або взагалі без підтягнення.

Вхід без підтяжки означає, що опір входу великий і будь-яка електрична перешкода може викликати появу на такому вході одиницю або нуль, причому зробити це не передбачувано. Щоб уникнути цього, потрібно використовувати підтягнення, вона дозволяє встановити на вході будь-який стійкий стан, що не залежатиме від перешкод. Підтягнення являє собою резистор великого опору, підключений одним кінцем до землі або до плюса живлення, а іншим кінцем – до входу. Наприклад, якщо підключене підтягнення до плюса живлення, то, коли вивід контролера нікуди не припаяний, на ньому завжди логічна одиниця. Якщо припаяти кнопку між цим виводом і землею, то кожного разу при натисканні кнопки на виводі буде з'являтися логічний нуль.

Якби підтягнення було вимкнене, то в момент натискання кнопки на виводі так само з'являвся б нуль, але при відпусканні кнопки вивід міг би легко «зловити» будь-яку перешкоду та викликати появу логічної одиниці на виводі. У результаті мікроконтролер вважав би, що хтось хаотично тисне на кнопку.

Для кращого розуміння вищевикладеного матеріалу наведено фрагменти коду налаштовування портів введення-виведення для різних завдань.

Приклад налаштовування портів на вивід для блимання світлодіодів:

```
// Структурні змінні для ініціалізації портів і налаштовування ліній переривання
GPIO_InitTypeDef gpioConf;
// Ініціалізація входу, підключеного до світлодіодів
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
```

gpioConf.GPIO_Mode = GPIO_Mode_OUT; // На вивод

gpioConf.GPIO_Speed = GPIO_Speed_100MHz; // Частота перемикання 100 МГц

gpioConf.GPIO_OType = GPIO_OType_PP; // Режим підтягування резисторами увімкнений

gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL; // Без підтягування

GPIO_Init (GPIOD, & gpioConf); // Ініціалізація налагоджувальної структури

Для налаштовування порту введення-виведення спочатку створюється змінна структури налаштування GPIO_InitTypeDef gpioConf. Далі вмикається порт D функцією RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE), де RCC_AHB1Periph_GPIOD – тактування порту D (рис. 4.1).

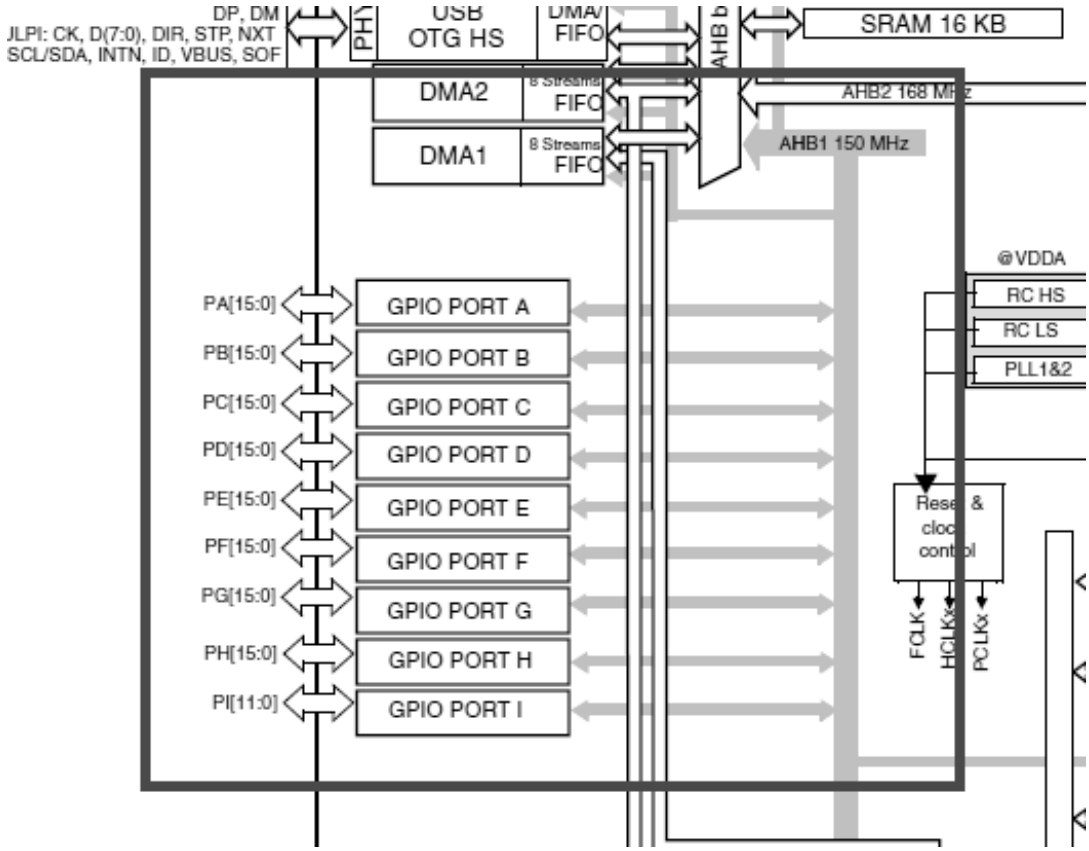


Рисунок 4.1 – Тактування порту D

З рис. 4.1 видно, що порт D отримує тактові імпульси від шини AHB1 150 MHz. Рядок коду gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15 дозволяє вибрати виводи порту (PD12-PD15), які потрібно налаштувати на виведення.

Команда gpioConf.GPIO_Mode = GPIO_Mode_OUT налаштовує виводи порту на виведення. Можливі інші варіанти: GPIO_Mode_IN – налаштовування виводу порту на вхід, GPIO_Mode_AF – налаштовування виводу порту на альтернативну функцію (використання його периферією, наприклад

UART), GPIO_Mode_AN – аналоговий режим (при використанні виводу як вхід АЦП). Частота перемикання виводу налаштовується таким чином: gpioConf.GPIO_Speed = GPIO_Speed_100MHz. Можуть бути такі варіанти: GPIO_Speed_2MHz, GPIO_Speed_25MHz, GPIO_Speed_50MHz, GPIO_Speed_100MHz. Відповідно частоту можна вибрати 2 МГц, 25 МГц, 50 МГц, 100 МГц.

Команда gpioConf.GPIO_OType = GPIO_OType_PP включає режим push-pull підтягнення виводу порту резисторами (можливий варіант без підтягнення виводу порту Open Drain – GPIO_OType_OD).

Рядок gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL налаштовує режим роботи порту без підтягнення (можливі варіанти GPIO_PuPd_UP, GPIO_PuPd_DOWN підтягнення до одиниці (високого потенціалу), нуля (землі), відповідно).

Команда GPIO_Init (GPIOD, & gpioConf) записує всі налаштування в спеціальні регістри для налаштовування портів, тобто, іншими словами, відбувається застосування всіх раніше обраних налаштувань.

Наведемо фрагмент коду налаштовування виводу порту на введення. Пін порт налаштовується на введення, якщо потрібно підключити до мікроконтролера будь-який двійковий датчик або кнопку.

```
GPIO_InitTypeDef gpioConf; // Змінна для налаштовування порту
// Ініціалізація входу, підключеного до кнопки
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE); // Увімкнути порт A
gpioConf.GPIO_Pin = GPIO_Pin_0; // Вибір виводу PA0
gpioConf.GPIO_Mode = GPIO_Mode_IN; // Налаштовування на вхід
GPIO_Init (GPIOA, & gpioConf); // Застосування налаштовань
```

Рядок gpioConf.GPIO_Mode = GPIO_Mode_IN дозволяє налаштувати вивід порту на введення.

Програмний код для налаштовування порту на альтернативну функцію для використання його виводів, наприклад інтерфейсом UART, наведено нижче:

```
GPIO_InitTypeDef gpioConf;
// Увімкнення портів і UART3
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE);
// Порти під UART3 задіяні
GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3);
GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3);
// Конфігуруємо UART TX UART RX як альтернативну функцію
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; // Ініціалізація і вхід і вихід як Alternate Function
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init (GPIOC, & GPIO_InitStructure);
```

Команда GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF налаштовує виводи порту С PC10 і PC11 на альтернативну функцію. Виводи підтягуються до одиниці (GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP). Команди GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3) і GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3) призначають виводи PC10, PC11 для UART3.

Функції RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE), RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE) вмикають порт С і UART3, відповідно. Частота перемикань виводів – 50 МГц.

Наведено налаштовування при використанні виводу в якості входу АЦП:

```
GPIO_InitTypeDef gpio; // Змінна для налаштовування
// Вмикаємо порт А
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);
// Режим аналогового входу
gpio.GPIO_Mode = GPIO_Mode_AN;
// Налаштовуємо вивід PA1
gpio.GPIO_Pin = GPIO_Pin_1;
// Застосовуємо налаштування
GPIO_Init (GPIOA, & gpio);
gpio.GPIO_Mode = GPIO_Mode_AN;
```

Команда gpio.GPIO_Mode = GPIO_Mode_AN налаштовує PA1 на аналоговове введення.

Таким чином, у цьому розділі було розглянуто призначення портів, основні режими їх роботи, коректне їх налаштування для різних завдань. Усі роз'яснення супроводжувалися прикладами практичного застосування налаштувань портів у програмному коді на мові С. Ці приклади можуть бути використані й модифіковані для власних потреб. Налаштування портів проводилося за допомогою стандартних бібліотек STM32F4.

Програмування з використанням стандартних бібліотек є гнучким, наочним і зрозумілішим, особливо для початківців. Однак усі операції можна здійснювати без використання стандартних бібліотек периферії. Можна безпосередньо заносити шістнадцятирічні або десяткові значення в спеціальні керівні регістри мікроконтролера. Однак для початківців у програмуванні краще використовувати стандартні бібліотеки, що й буде зроблено при виконанні циклу практичних робіт з вивчення програмування STM32F4Discovery.

РОЗДІЛ 5
ПОЧАТОК РОБОТИ З НАЛАГОДЖУВАЛЬНОЮ ПЛАТОЮ STM32F4DISCOVERY

5.1 Підключення та встановлення середовища розроблення

Розділ призначений для тих, хто до цього моменту не мав ні практичних, ні теоретичних знань щодо роботи з мікроконтролерами, а також для тих, кому наявних знань недостатньо для початку роботи з платою. Незважаючи на зовнішню складність плати, її будову добре продумано, тому навіть абсолютним новачкам у цій справі не складе труднощів опанувати основи роботи з STM32F4Discovery.

5.2 Апаратне забезпечення для підключення налагоджувальної плати STM32F4Discovery

Єдине, що потрібно для підключення плати до ПК, – це кабель USB з рознімами USB типу A і mini-USB типу B, а також кабель USB типу A і micro-USB типу B (для налаштування). Для прошивання мікроконтролера можна використовувати тільки mini-USB (рис. 5.1). У комплекті поставки такий шнур відсутній. Ці кабелі знайомі завдяки телефонам, планшетам або фотоапаратам, які часто підключаються до комп'ютера саме за їх допомогою.

Рисунок 5.1 – Кабель mini-USB

При підключенні будьте уважні: на платі є два рознімами USB, один з яких призначений для підключення, а другий – для реалізації роботи з USB.

Передбачається, що на комп'ютері встановлено операційну систему сімейства Windows (XP, Vista, Windows 7 тощо). Існує можливість підключення до ПК з Linux, але це вимагатиме набагато більше зусиль зі встановлення програмного забезпечення.

5.3 Перше підключення налагоджувальної плати до ПК

Для роботи з платою необхідно, щоб відбулося встановлення драйверів. Спочатку потрібно завантажити й встановити ST-Link USB-driver (рис. 5.2). Для встановлення достатньо запустити виконуваний файл завантаженого драйвера та додержуватися інструкцій на екрані. Слід також завантажити й встановити програму STM32 ST-Link Utility (рис. 5.3).

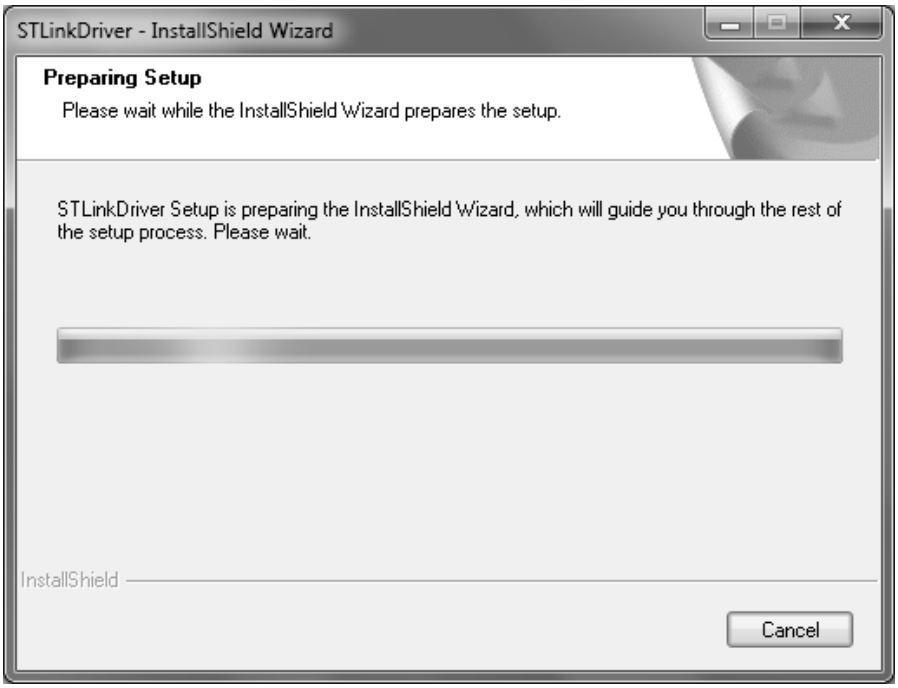


Рисунок 5.2 – Майстер встановлення програми STLinkdriver

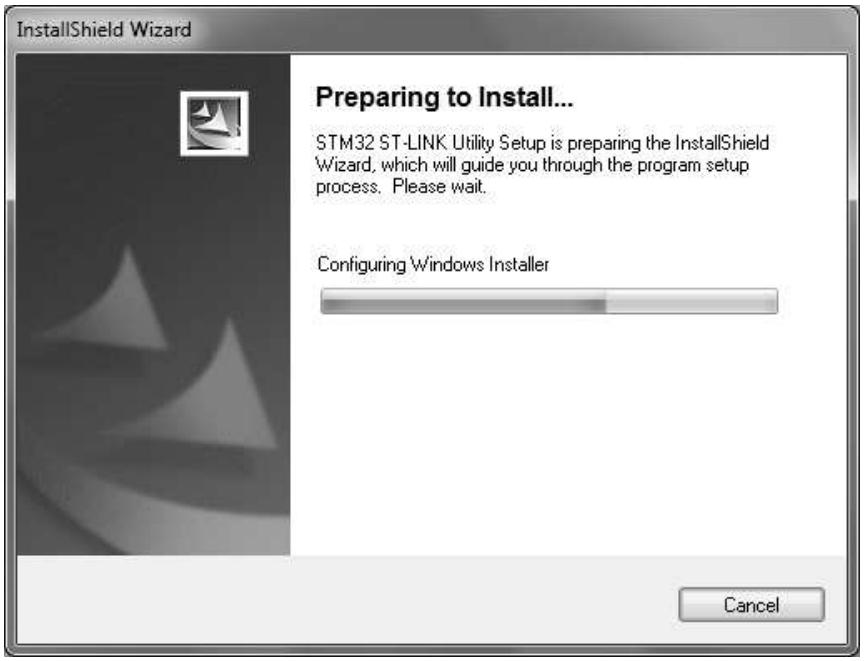


Рисунок 5.3 – Майстер встановлення утиліти STLink

Процес встановлення драйверів при підключенні пристрою буде розглянутий на прикладі операційної системи Windows 7.

При підключенні пристрою відразу ж починається встановлення драйверів для нового пристрою. В області системного трея з'являється повідомлення такого вигляду (рис. 5.4):

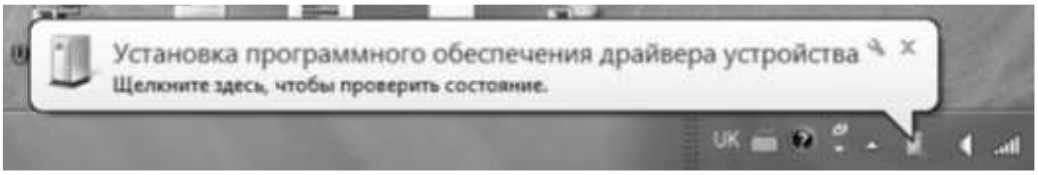


Рисунок 5.4 – Повідомлення при підключенні пристрою

Натискання на повідомленні лівою кнопкою мишки відкриває вікно, яке відображає динаміку встановлення драйверів (рис. 5.5).

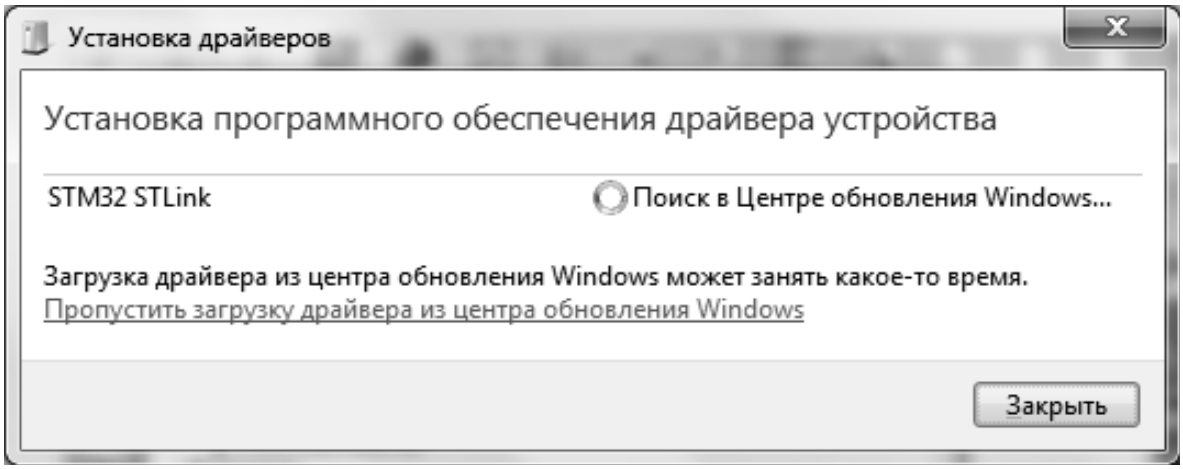


Рисунок 5.5 – Динаміка встановлення драйвера

У результаті успішного встановлення з'являється повідомлення, у якому зазначено ім'я підключеного пристрою – STMicroelectronics STLink Dongle (рис. 5.6).

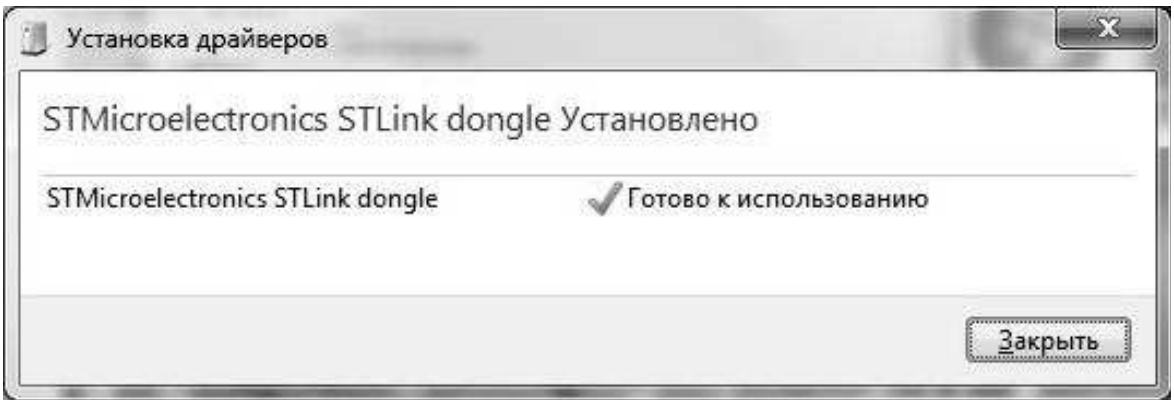


Рисунок 5.6 – Повідомлення при успішному встановленні

У диспетчері пристроїв (Пуск / Комп'ютер / Властивості / Диспетчер пристроїв) можна знайти підключений пристрій (рис. 5.7).

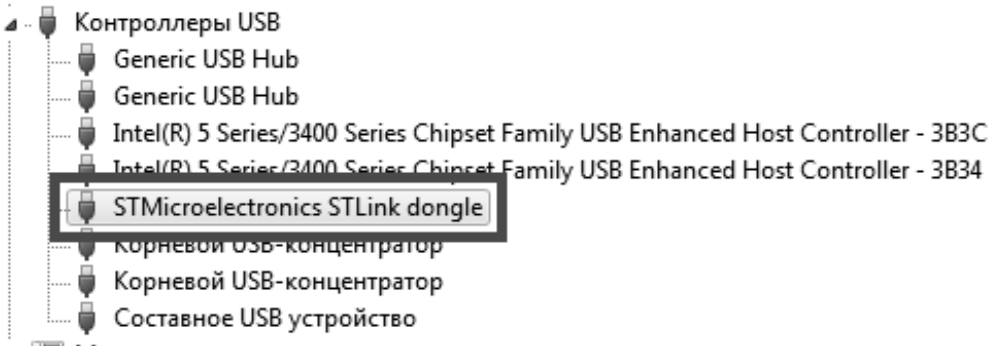


Рисунок 5.7 – Пошук підключеного пристрою в диспетчері пристроїв

Тепер пристрій готовий для взаємодії з програмним забезпеченням, встановленим на ПК.

5.4 Програмні засоби для прошивання плати

Незважаючи на те, що середовища розробки інтегрують в собі функціональність щодо прошивання мікроконтролерів, корисно буде знати, що існують і окремі програми, призначені спеціально для роботи з пам'яттю (програмування, очищення, перевірка) пристроїв. Крім того, при роботі з режимами зниженого енергоспоживання саме ці програми дозволяють виконати програмування незалежно від поточного режиму роботи, з чим можуть виникнути проблеми при виконанні цих дій із середовища розроблення.

В цьому підрозділі розглянуто два програмні продукти, які реалізують функціональність для виконання прошивання без використання середовища розробки: STM32 ST-LINK Utility і ST Visual Programmer.

Програма STM32 ST-LINK Utility призначена для роботи з 32-розрядними контролерами через інтерфейс ST-LINK (у тому числі і з його другою версією). Її інтерфейс (рис. 5.8) організований максимально просто й зрозуміло. Основну робочу область організовано у вигляді двох вкладок, у яких відображається пам'ять пристрою, а також наведений двійковий файл для запису. Найбільш важливі команди зосереджені в меню Target. Якщо в момент відкриття програми пристрій не було підключено до ПК, то виконати підключення слід командою Target / Connect. Після цього в панелі повідомлень повинно з'явитися повідомлення про успішне підключення й інформація про приєднаний пристрій. Виконати програмування пристрою можна командою Target / Program .... При цьому слід заздалегідь відкрити файл з програмою. Стирання пам'яті програм пристрою виконується командою Target / Erase Chip.

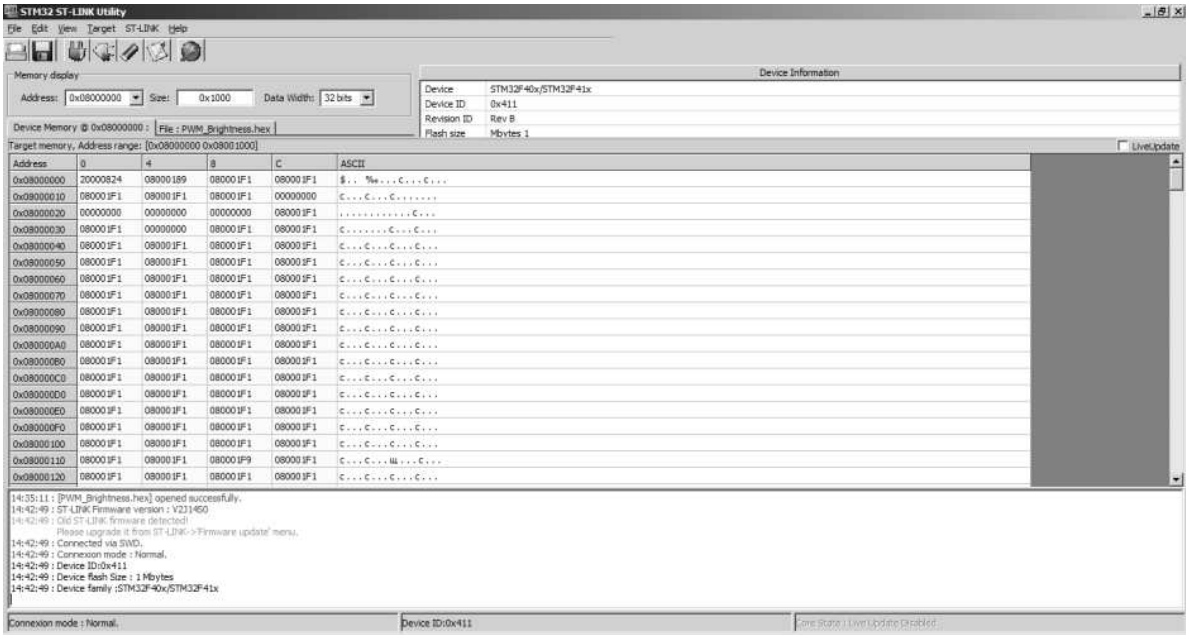


Рисунок 5.8 – Вікно програми STM32 ST-LINK Utility

Програма ST Visual Programmer (STVP) є універсальним засобом для прошивання мікроконтролерів виробництва STMicroelectronics.

Вона може працювати з пристроями сімейств STM32, STM8, а також STM7 і з великою кількістю інтерфейсів, що також є перевагою. В іншому разі STVP і STM32 ST-LINK Utility дуже схожі між собою як за інтерфейсом (рис. 5.9), так і за функціональністю.

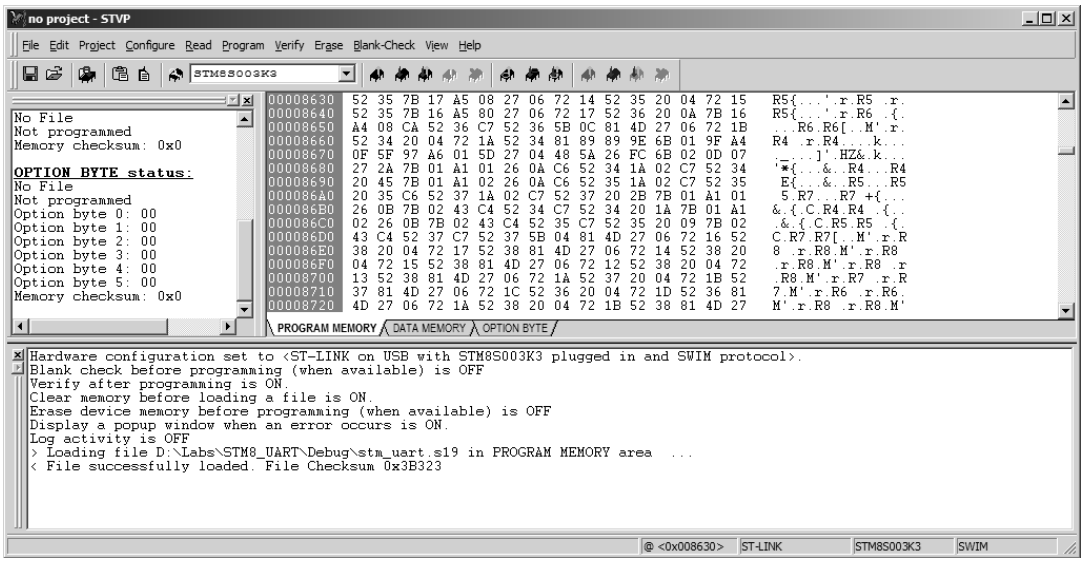


Рисунок 5.9 – Вікно програми STVP

Використання STVP багато в чому аналогічне використанню *STM32 ST-LINK Utility*, але необхідно самостійно задати налаштування підключення. Ці дії виконуються в діалоговому вікні (рис. 5.10), яке відкривається командою Configure / Configure ST Visual Programmer. У ньому вказується апаратне забезпечення, яке використовується для підключення, і пристрій.

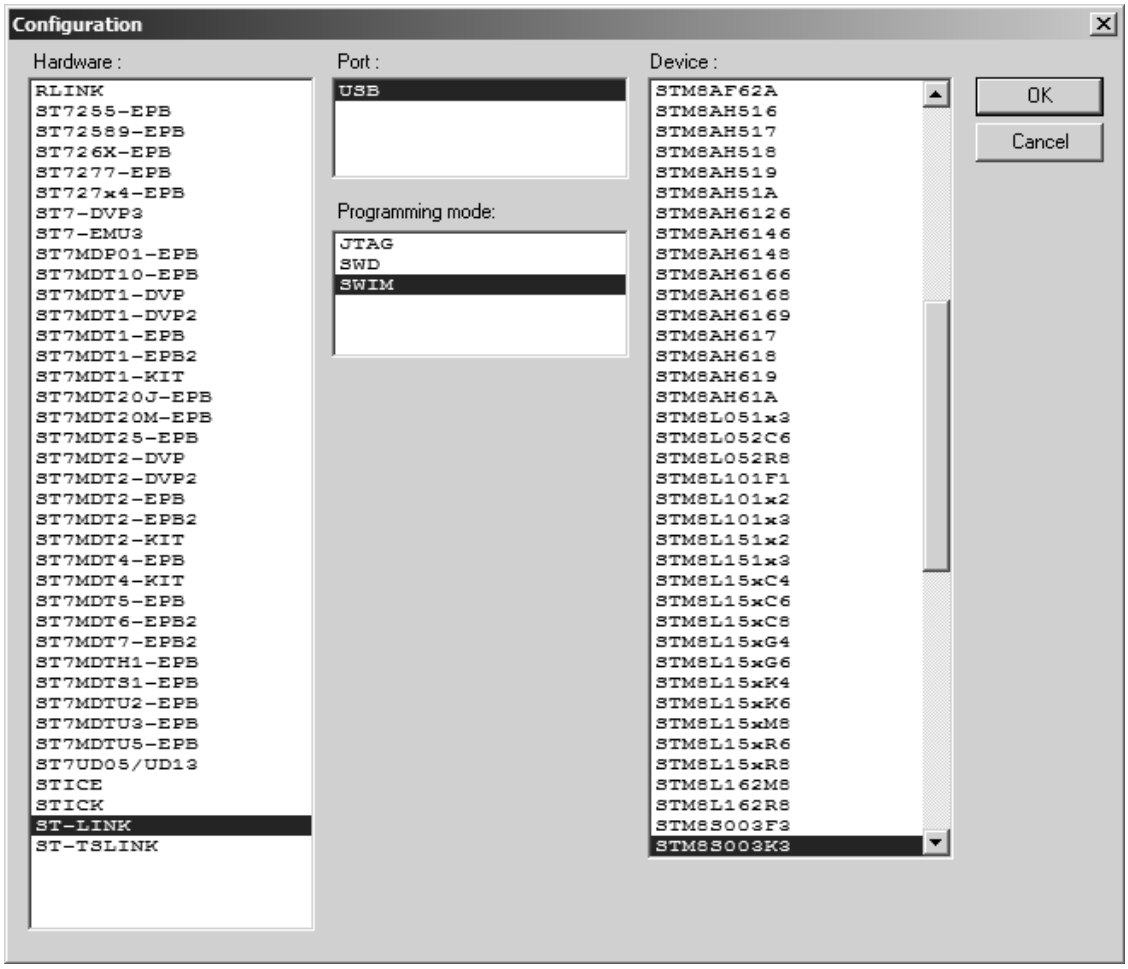


Рисунок 5.10 – Вікно налаштування програмування пристрою

Робота з налагоджувальною платою STM32F4Discovery з використанням описаних інструментів відбувається за схожим сценарієм, тому перевага в їх використанні залежить від користувача. Автоматичне підключення при запуску полегшить роботу новачків. Для тих, хто працює з різними пристроями, можна рекомендувати STVP.

Засоби налаштування частоти роботи мікроконтролера

За умовчанням робоча частота мікроконтролера налагоджувальної плати становить 16 МГц. Цю частоту забезпечує внутрішній високочастотний генератор тактових імпульсів (High Speed Internal Oscillator – HSI). Однак максимальна робоча частота становить 168 МГц, що значно більше від значення за умовчанням. Для того, щоб налаштувати роботу пристрою на потрібній частоті, виробник пропонує скористатися спеціально розробленим програмним забезпеченням – Clock Configuration Tool. Воно являє собою xls-файл з макросами, тому для його використання слід задати дозвіл виконання макросів у Excel. Інтерфейс програми (рис. 5.11) організований на одному аркуші з використанням полів введення й вибору значень і наочно відображає схему тактування, яка використовується в пристрої.

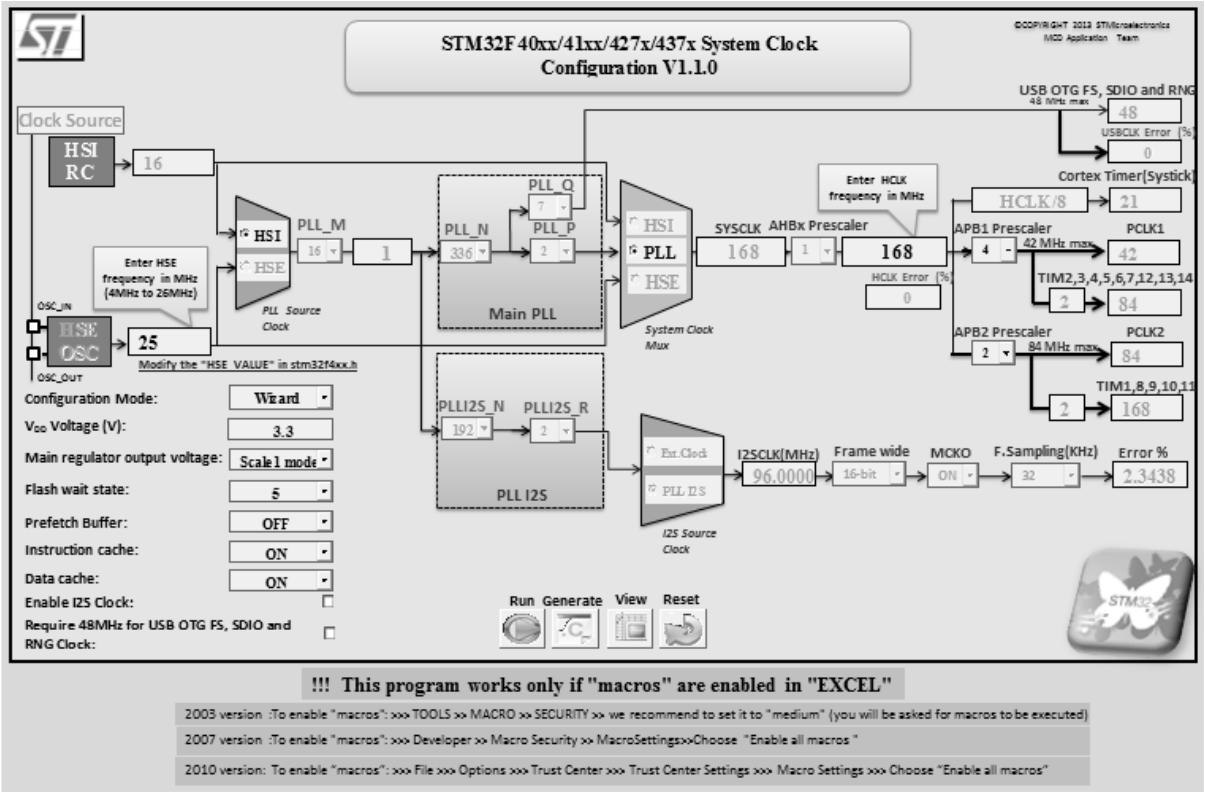


Рисунок 5.11 – Інтерфейс програми для налаштування системи тактування

У нижній частині схеми знаходяться кнопки, за допомогою яких запускаються макроси, що виконують основну роботу. При натисканні на кнопку Generate в папці з файлом генерується новий файл конфігурації з ім'ям system_stm32f4xx.c. Його можна підключити до проекту в середовищі розробки, замінивши існуючий файл. Для того, щоб застосувати налаштування, слід відстежити, чи викликається функція SystemInit(), оскільки без її виклику пристрій буде працювати на частоті за умовчанням. У разі відсутності виклику функції її слід викликати самостійно, наприклад, в основній функції main. Після цього пристрій буде працювати на тій частоті, яка задана, тому, можливо, слід змінити значення параметрів роботи пристроїв для підтримання коректної роботи в подальшому.

РОЗДІЛ 6
ЕТАПИ ПРАКТИЧНОГО ВИКОРИСТАННЯ НАЛАГОДЖУВАЛЬНОЇ ПЛАТИ

6.1 Створення проекту в середовищі Atollic True Studio й ознайомлення з портами введення-виведення мікроконтролера

Метою цього підрозділу є ознайомлення із середовищем розробки Atollic True Studio, створення нового проекту. Ознайомлення з портами введення-виведення STM32F4Discovery. Обладнання й програмне забезпечення, що використовується при цьому, включає в себе плату STM32F4 Discovery, середовище розробки Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite відповідно.

Створення нового проекту в Atollic True Studio

Новий проект буде створюватися для мікроконтролера STM32F4.

Вибирається директорія, де буде зберігатися майбутній проект (можна створити порожню папку, наприклад C: \ MyF4). Далі запускається Atollic True Studio й вибирається раніше створена папка як робоча (рис. 6.1).

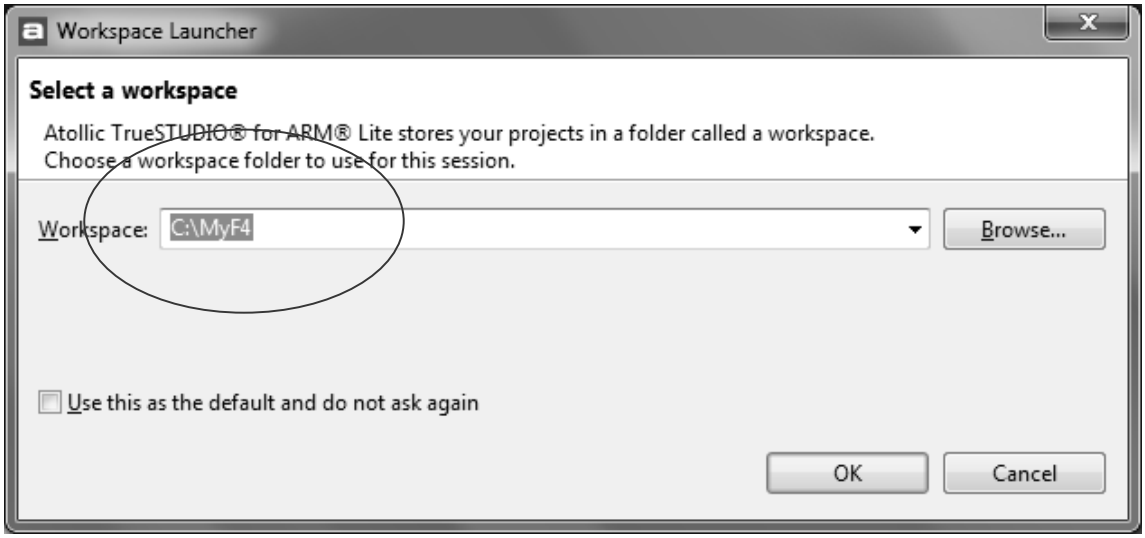


Рисунок 6.1 – Вибір робочої директорії нового проекту

При натисканні ОК завантажується інтегроване середовище розроблення Atollic True Studio (рис. 6.2).

Для створення нового проекту вибирається меню File – Create new C Project. З'явиться діалогове вікно введення імені майбутнього проекту. Далі вводиться ім'я проекту, наприклад MyFirstProject (рис. 6.3)

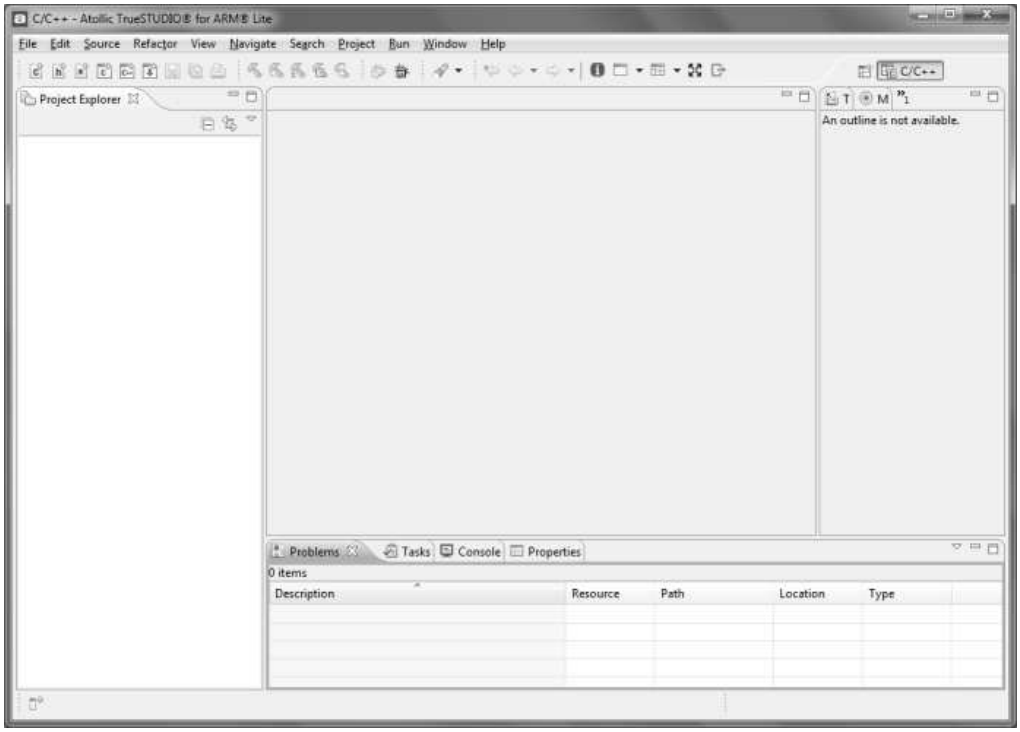


Рисунок 6.2 – Середовище розроблення Atollic True Studio при завантаженні

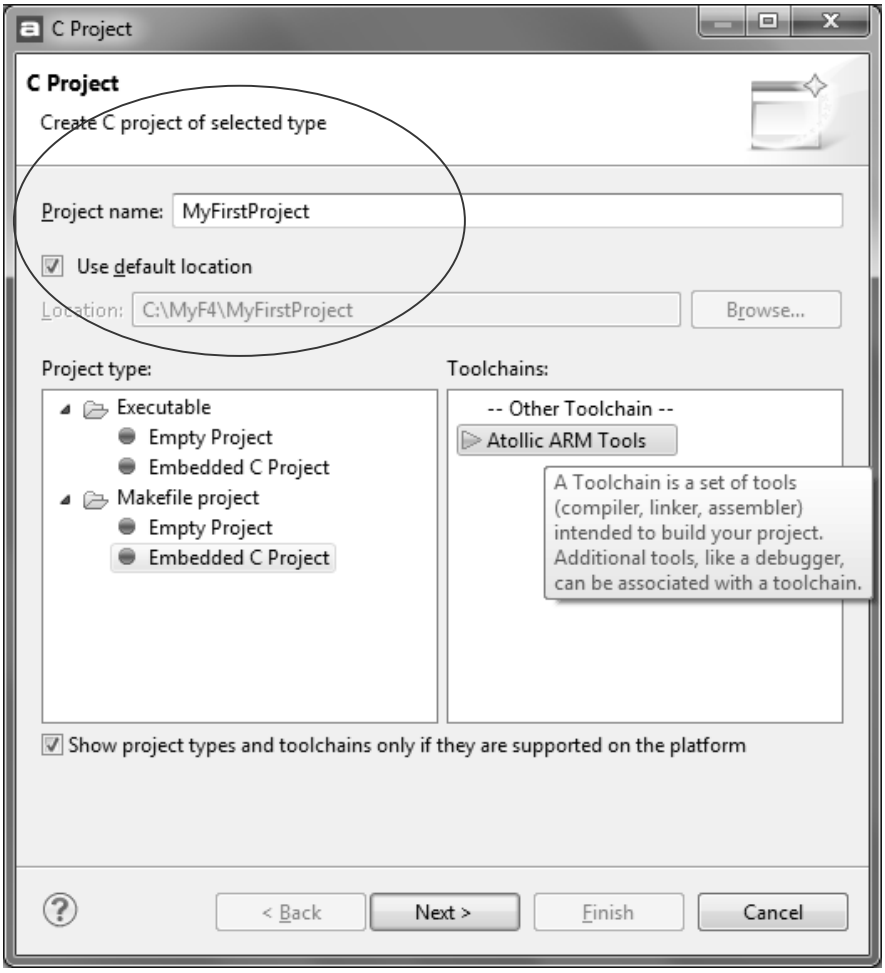


Рисунок 6.3 – Діалогове вікно вибору імені проекту

Після введення імені проекту потрібно натиснути Next для вибору виробника мікроконтролерів, типу налагоджувальної плати тощо. Для STM32F4Discovery потрібно встановити всі налаштування відповідно до рис. 6.4.

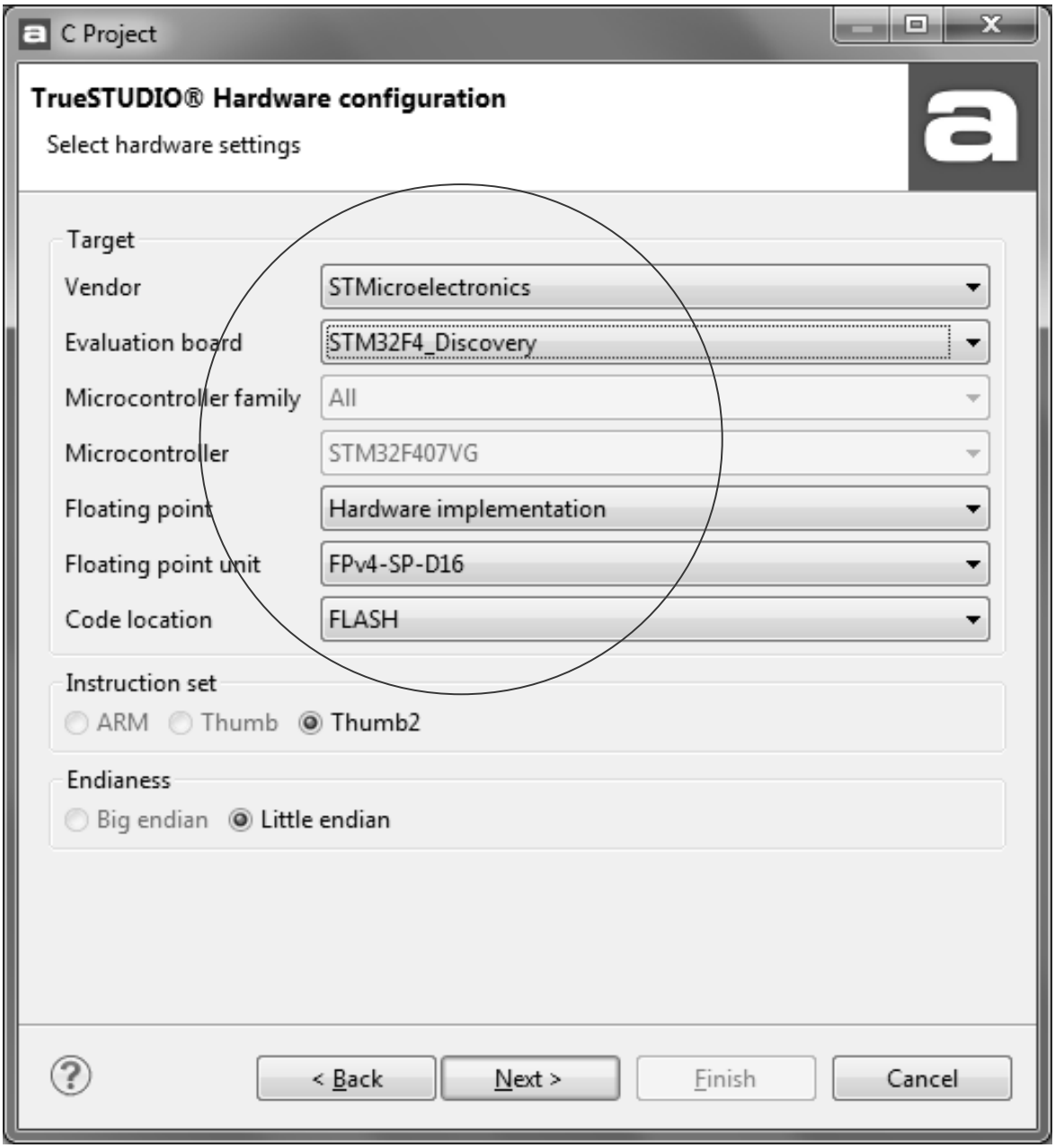


Рисунок 6.4 – Вибір використовуваної налагоджувальної плати

Далі – Next. З'явиться діалогове вікно, зображене на рис. 6.5. Усі налаштування – за умовчанням.

Після потрібно натиснути Next. З'явиться наступне діалогове вікно (рис. 6.6).

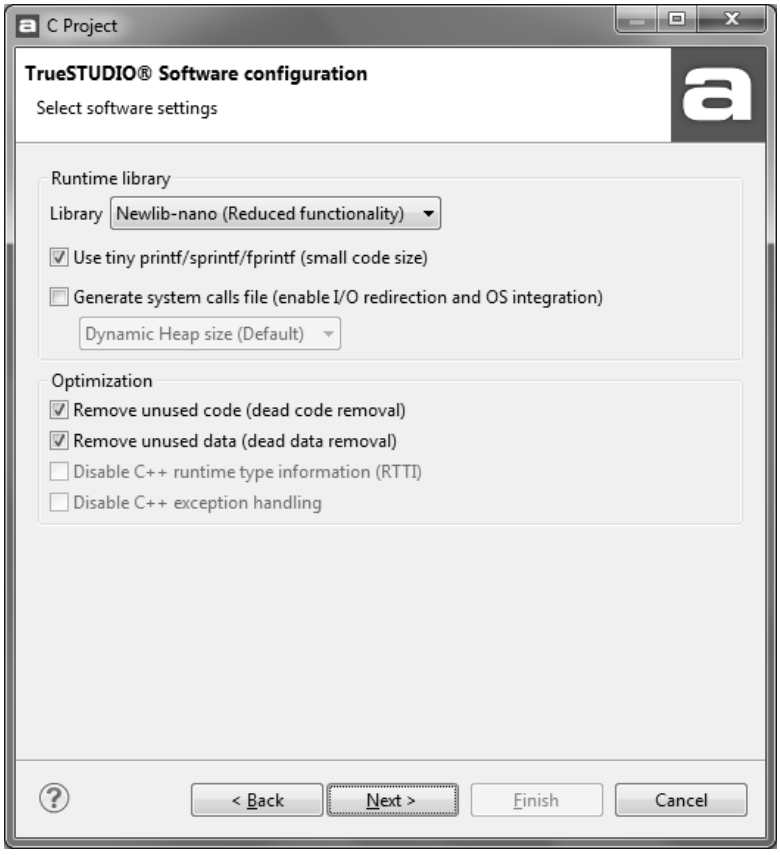


Рисунок 6.5 – Діалогове вікно вибору налаштовань програмного забезпечення

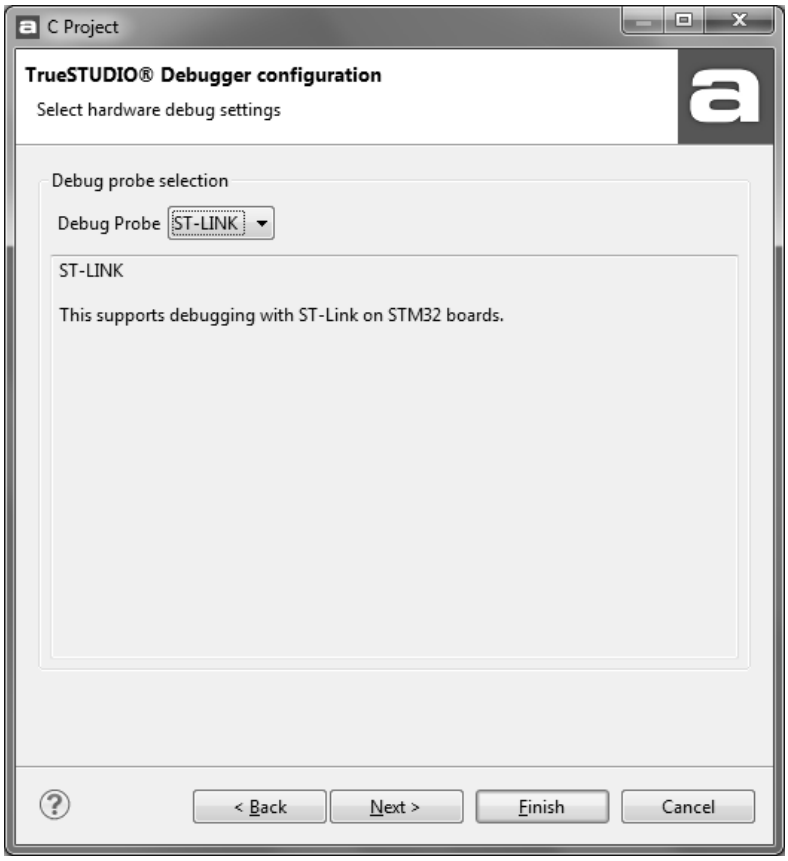


Рисунок 6.6 – Вибір апаратного налаштовувача

Натиснути клавішу Finish. Запускається процес створення шаблону демонстраційного проекту. Далі буде проведено його компіляцію. Проект матиме вигляд, наведений на рис. 6.7.

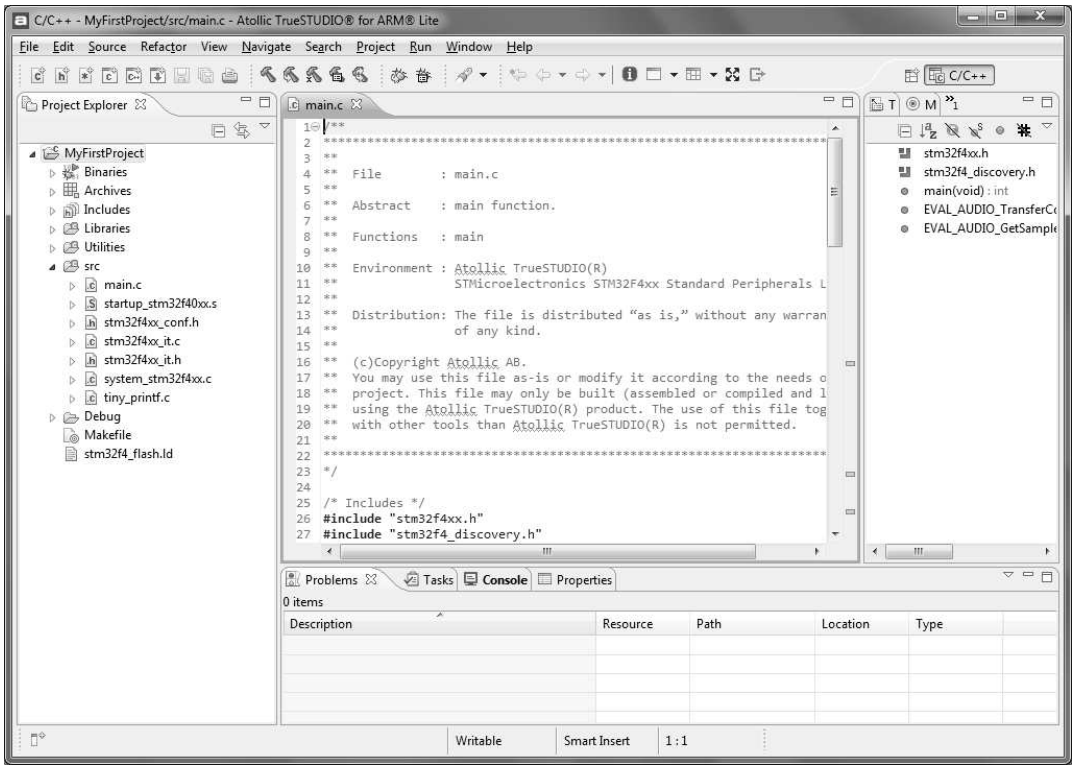


Рисунок 6.7 – Демонстраційний проект, створений Atollic True Studio

Код цього проекту можна змінювати для реалізації різних завдань за допомогою STM32F4Discovery.

Примітка: якщо змінений проект не компілюватиметься, можна видалити невикористовувані бібліотеки (рис. 6.8), після цього проект повинен успішно компілюватися.

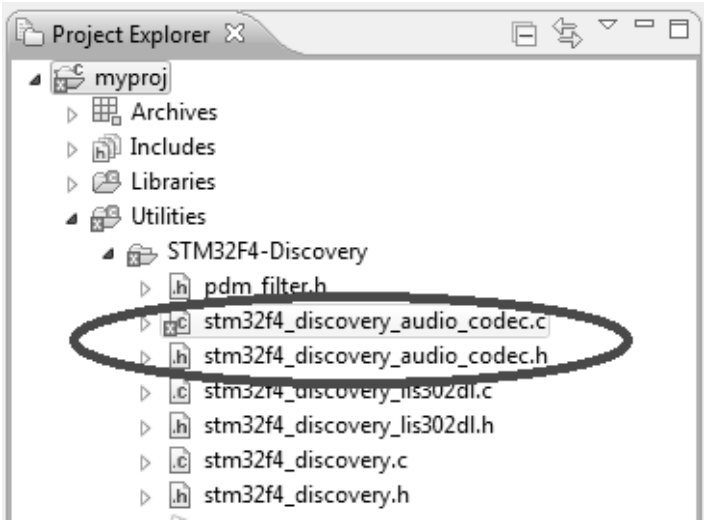


Рисунок 6.8 – Бібліотеки, які не використовуються у рамках цієї монографії

Основні положення

Мікроконтролер STM32F407VG містить п'ять 16-розрядних портів введення-виведення загального призначення, які позначені як GPIOx, де x може мати значення A, B, C, D, E. Кожен порт GPIO має чотири 32-бітних регістра конфігурації (GPIOx_MODER, GPIOx_TYPER, GPIOx_SPEEDR , GPIOx_ORD), два 32-бітових регістри даних (GPIOx_ODR, GPIOx_IDR) і два 32-бітових регістри вибору додаткових функцій (GPIOx_AFRH і GPIOx_AFRL). Світлодіоди, призначені для програмування, на платі підключені до порту D (рис. 6.9) [8].

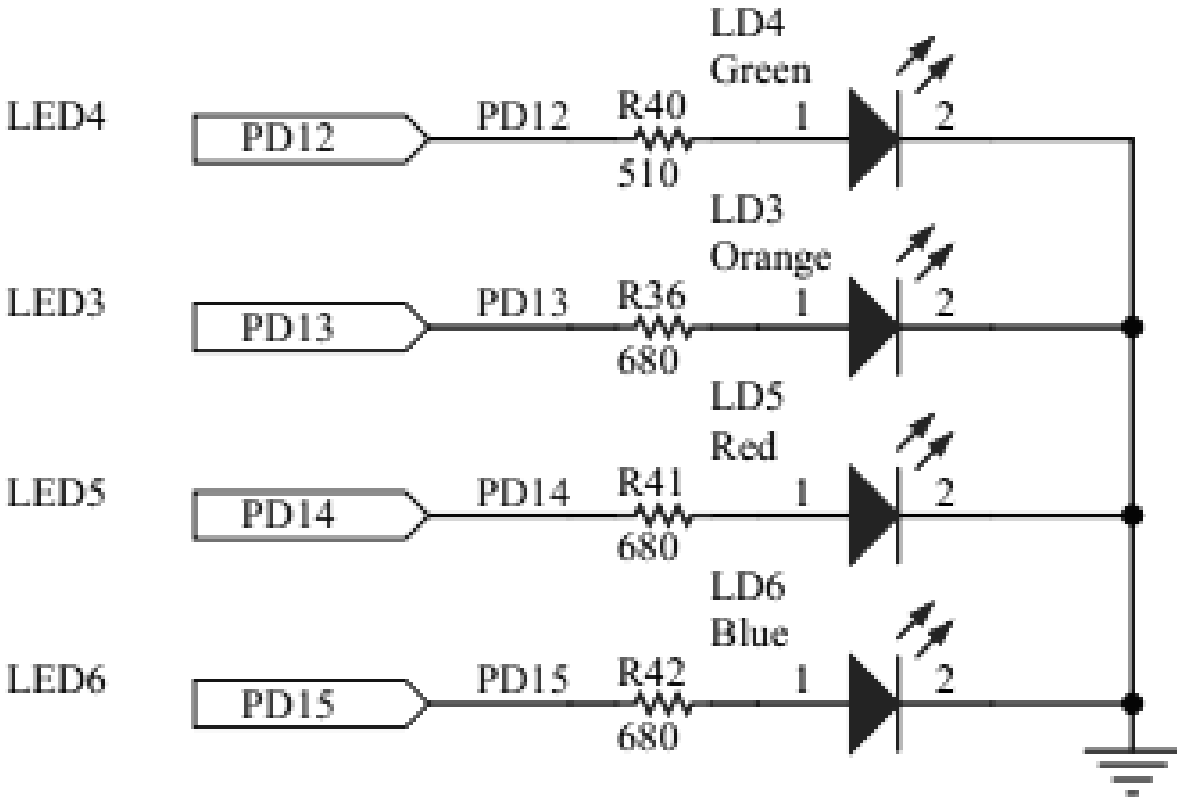


Рисунок 6.9 – Схема підключення світлодіодів до порту на платі

Приклад програми

Розглянуто приклад програми, яка запалює, а потім гасить через певний час світлодіод LED3 Orange на платі STM32F4Discovery.

Цей світлодіод підключений до порту PD13. Фрагмент опису портів введення-виведення мікроконтролера зображено на рис. 6.10.

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	US302DL or US3DSH	Push button	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PD8	FSMC_D13/ USART3_TX	55	-	-	-	-	-	-	-	-	-	-	-	-	40	-
PD9	FSMC_D14/ USART3_RX	56	-	-	-	-	-	-	-	-	-	-	-	-	41	-
PD10	FSMC_D15/ USART3_CK	57	-	-	-	-	-	-	-	-	-	-	-	-	42	-
PD11	FSMC_A16/ USART3_CTS	58	-	-	-	-	-	-	-	-	-	-	-	-	43	-
PD12	FSMC_A17/ TIM4_CH1/ USART3_RTS	59	-	-	-	-	GREEN	-	-	-	-	-	-	-	44	-
PD13	FSMC_A18/ TIM4_CH2	60	-	-	-	-	ORANGE	-	-	-	-	-	-	-	45	-
PD14	FSMC_D0/ TIM4_CH3	61	-	-	-	-	RED	-	-	-	-	-	-	-	46	-
PD15	FSMC_D1/ TIM4_CH4	62	-	-	-	-	BLUE	-	-	-	-	-	-	-	47	-
PE0	TIM4_ETR/ FSMC_NBL0/ DCMI_D2	97	-	-	INT1	-	-	-	-	-	-	-	-	-	-	17
PE1	FSMC_NBL1/ DCMI_D3	98	-	-	INT2	-	-	-	-	-	-	-	-	-	-	18
PE2	TRACECLK/ FSMC_A23/ ETH_MII_TXD3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	15
PE3	TRACED0/ FSMC_A19	2	-	-	CS_I2C/SPI	-	-	-	-	-	-	-	-	-	-	16

Рисунок 6.10 – Опис портів введення-виведення

Лістинг програми

```
// Підключаються бібліотеки
#include "Stm32f4xx.h"
#include "Stm32f4xx_rcc.h"
#include "Stm32f4xx_gpio.h"

// Затримання
```

```
void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}

int main (void)

{

GPIO_InitTypeDef gpioConf;

// Ініціалізація входу, підключеного до світлодіода

 gpioConf.GPIO_Pin = GPIO_Pin_13;
 gpioConf.GPIO_Mode = GPIO_Mode_OUT;
 gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
 gpioConf.GPIO_OType = GPIO_OType_PP;
 gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init (GPIOD, & gpioConf);

while(1) {

// Запалюємо і гасимо світлодіод із затриманням

  GPIO_SetBits (GPIOD, GPIO_Pin_13);
  Delay (5000);
  GPIO_ResetBits (GPIOD, GPIO_Pin_13);
  Delay (5000);

}
}
```

Порядок роботи

1. На основі коду прикладу програми створити свій проект у середовищі розробки й перевірити його працездатність.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки й у режимі налаштування.

3. Створення нового проекту в середовищі розробки при виконанні можливих варіантів завдання, наведених у табл. 6.1.

4. Реалізація необхідної функціональності.

5. Прошивання плати й демонстрація роботи програми.

6. Аналіз і оцінка створеного проекту.

Можливі комбінації варіантів схем увімкнення світлодіодів

У табл. 6.1 наведено можливі комбінації схем увімкнення світлодіодів, розташованих на налагоджувальній платі:

Таблиця 6.1 – Варіанти увімкнення світлодіодів

Варіант	LD4	LD3	LD5	LD6
1	1	2	3	4
2	3	2	1	4
3	4	3	2	1
4	1	3	2	4
5	-	3	1	2
6	1	-	4	2
7	4	1	-	3
8	1	2	4	3
9	3	4	1	2
10	1	3	2	4
11	2	4	4	1
12	3	2	1	4
13	4	1	2	3
14	1	4	3	2
15	2	3	1	4
16	3	2	4	1
17	4	1	2	3
18	1	3	2	4
19	4	3	1	2
20	3	1	2	4

Номери увімкнення світлодіодів привласнюються за їх номерами в складі порту D (можна знайти в документації).

6.2 Використання переривань

Метою цього підрозділу є ознайомлення з призначенням переривань мікроконтролера STM32F4 і його основними можливостями. Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite.

Основні відомості

Переривання – механізм, який дозволяє апаратному забезпеченню повідомляти про настання важливих подій у своїй роботі. У момент, коли відбувається переривання, процесор перемикається з виконання основної

програми на виконання відповідного оброблювача переривань. Як тільки виконання оброблювача завершено, триває виконання основної програми з місця, у якому вона була перервана.

Для використання переривань необхідно спочатку налаштувати регістр, який називається Nested Vector Interrupt Controller (NVIC), вбудований контролер вектора переривань. Цей регістр є стандартною частиною архітектури ARM і зустрічається на всіх процесорах, незалежно від виробника. NVIC розроблений таким чином, що затримання переривання було мінімальним. NVIC підтримує вбудовані переривання з 16 рівнями пріоритету.

У мікроконтролері STM32F4 є 22 лінії переривання (EXTI0-EXTI22). Перші 0–15 можуть використовуватися певними портами введення-виведення (див. документацію). Решта 16–22 можуть бути використані для інших цілей.

У цьому прикладі буде використовуватися лінія переривання EXTI0. Призначена для користувача кнопка підключена до порту PA0, згідно з документацією цей порт може використовувати лінію переривання EXTI0 (рис. 6.11) [9].

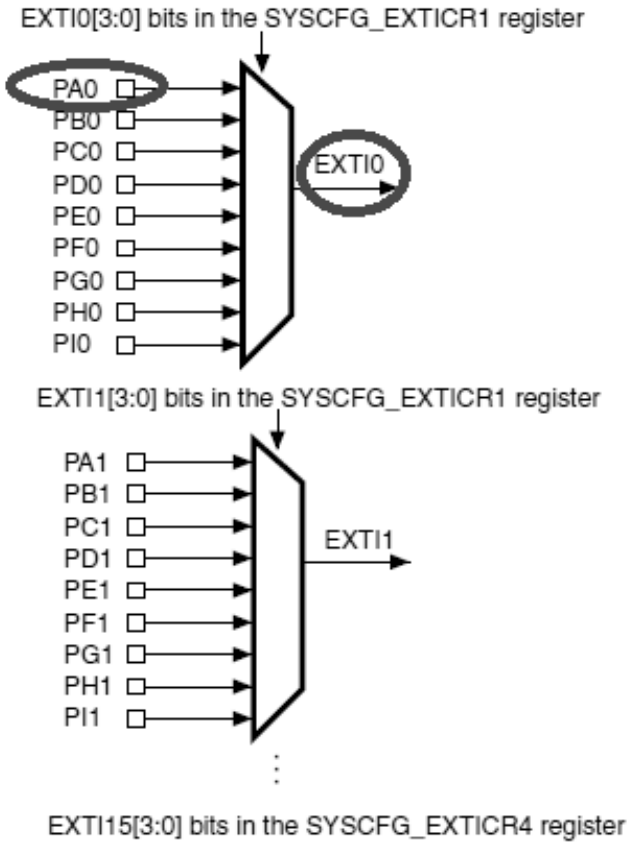


Рисунок 6.11 – Діаграма ліній переривань

Приклад програми

Розглянуто варіант генерації переривань при натисканні користувальницької кнопки на налагоджувальній платі STM32F4Discovery. Наведено приклад програми, де при натисненні користувальницької кнопки буде

змінюватися напрямок бігу вогню, реалізованого на світлодіодах LD4, LD3, LD5, LD6 налагоджувальної плати.

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_rcc.h"
#include "Stm32f4xx_gpio.h"

// i –перемикання режиму загоряння світлодіодів (вперед або назад)
// ind – додаткова змінна (4 світлодіоди) повинна бути не більше 4

int i = 1, ind= 1;

void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}
int main (void)
{
// Структурні змінні для ініціалізації портів і налаштування ліній переривання
GPIO_InitTypeDef gpioConf;
EXTI_InitTypeDef EXTI_InitStructure;

// Ініціалізація входу, підключеного до кнопки
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

// Включаємо тактування SYSCFG clock для вибору лінії переривання EXTI_0 (PA0)
RCC_APB2PeriphClockCmd (RCC_APB2Periph_SYSCFG, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_0;
gpioConf.GPIO_Mode = GPIO_Mode_IN;
GPIO_Init (GPIOA, & gpioConf);

// Ініціалізація входу, підключеного до світлодіодів
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
gpioConf.GPIO_Mode = GPIO_Mode_OUT;
gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
gpioConf.GPIO_OType = GPIO_OType_PP;
gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpioConf);
```

```
// Налаштування лінії переривання
NVIC_InitTypeDef nvic_struct;
nvic_struct.NVIC_IRQChannel = EXTI0_IRQn;
nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
nvic_struct.NVIC_IRQChannelSubPriority = 1;
nvic_struct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init (& nvic_struct);

// Переривання від PA0 встановлені
SYSCFG_EXTILineConfig (EXTI_PortSourceGPIOA,
EXTI_PinSource0);

// Конфігурація переривання від PA0
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;

//Переривання на PA0 по наростальному фронту
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init (& EXTI_InitStructure);

// Вмикаємо переривання
NVIC_EnableIRQ (EXTI0_IRQn);

while(1) {

// Перемикаємо напрямок загоряння світлодіодів
switch(Ind)
{
case 1: {
        GPIO_SetBits (GPIOD, GPIO_Pin_12);
        GPIO_ResetBits (GPIOD, GPIO_Pin_13);
        GPIO_ResetBits (GPIOD, GPIO_Pin_14);
        GPIO_ResetBits (GPIOD, GPIO_Pin_15);
        Delay (5000000);
        break;
}
case 2: {
        GPIO_SetBits (GPIOD, GPIO_Pin_13);
        GPIO_ResetBits (GPIOD, GPIO_Pin_12);
        GPIO_ResetBits (GPIOD, GPIO_Pin_14);
        GPIO_ResetBits (GPIOD, GPIO_Pin_15);
        Delay (5000000);
        break;
}
```

```
case 3: {
        GPIO_SetBits (GPIOD, GPIO_Pin_14);
        GPIO_ResetBits (GPIOD, GPIO_Pin_12);
        GPIO_ResetBits (GPIOD, GPIO_Pin_13);
        GPIO_ResetBits (GPIOD, GPIO_Pin_15);
        Delay (5000000);
        break;
}
case 4: {
        GPIO_SetBits (GPIOD, GPIO_Pin_15);
        GPIO_ResetBits (GPIOD, GPIO_Pin_13);
        GPIO_ResetBits (GPIOD, GPIO_Pin_14);
        GPIO_ResetBits (GPIOD, GPIO_Pin_12);
        Delay (5000000);
        break;
}
}

if(I == 1) {

if(ind> 4) {ind= 1;} else {ind++;}
        }

        else if (i == 2) {if (ind <0) {ind = 4;} else {ind--;}}

}
}

// Оброблювач переривання EXTI0
void EXTI0_IRQHandler (void)

{
        EXTI_ClearITPendingBit (EXTI_Line0); // Очищаємо прапорець
EXTI_Line0 - вхід PA0

        // Вперед або назад

        if(I == 1) {i = 2;}
        else i = 1;

}
```

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення і перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій за допомогою перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки і в режимі налагодження.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати і перевірка роботи програми.

6. Аналіз отриманих результатів.

Можливі варіанти реалізації переривань

На основі наведеного прикладу можливе здійснення власних налаштовань відповідно до табл. 6.2.

Таблиця 6.2 – Можливі варіанти реалізації переривань

№ з/п	Можливі варіанти
1	2
1	Реалізувати «біжний» вогонь на світлодіодах LD4, LD3, LD5, LD6. При натисканні користувальницької клавіші запускати / зупиняти «біжний» вогонь.
2	Реалізувати «біжну» тінь на світлодіодах LD4, LD3, LD5, LD6. При натисканні користувальницької клавіші запускати / зупиняти «біжну» тінь.
3	Увімкнути світлодіод LD4. При натисканні користувальницької клавіші змінювати частоту мерехтіння світлодіода.
4	При натисканні користувальницької клавіші вмикати світлодіод LD5, вимикати LD4. При повторному натисканні – навпаки.
5	Увімкнути світлодіод LD5, LD4. При натисканні користувальницької клавіші вимкнути ці світлодіоди. При повторному натисканні – увімкнути світлодіоди.
6	Увімкнути світлодіод LD5. При натисканні клавіші – вимкнути. При повторному натисканні –знову ввімкнути.
7	Увімкнути світлодіод LD4. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
8	Увімкнути світлодіод LD3. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
9	Увімкнути світлодіод LD6. При натисканні клавіші – вимкнути. При повторному натисканні знову – ввімкнути.
10	Реалізувати «біжний» вогонь на світлодіодах LD4, LD3. При натисканні користувальницької клавіші запускати / зупиняти «біжний» вогонь.

Продовження таблиці 6.2

1	2
11	Реалізувати «біжний» вогонь на світлодіодах LD5, LD6. При натисканні користувальницької клавіші запускати / зупиняти «біжний» вогонь.
12	Реалізувати «біжну» тінь на світлодіодах LD5, LD6. При натисканні користувальницької клавіші запускати / зупиняти «біжну» тінь.
13	Реалізувати «біжну» тінь на світлодіодах LD4, LD3. При натисканні користувальницької клавіші запускати / зупиняти «біжну» тінь.
14	Реалізувати «біжну» тінь на світлодіодах LD4, LD3, LD5. При натисканні користувальницької клавіші запускати / зупиняти «біжну» тінь.
15	Реалізувати «біжну» тінь на світлодіодах LD4, LD3, LD5. При натисканні користувальницької клавіші запускати / зупиняти «біжну» тінь.
16	Увімкнути світлодіоди LD4, LD3, LD5, LD6. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
17	Увімкнути світлодіоди LD4, LD3, LD5. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
18	Увімкнути світлодіоди LD4, LD5, LD6. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
19	Увімкнути світлодіоди LD5, LD4. При натисканні клавіші – вимкнути. При повторному натисканні – знову ввімкнути.
20	Увімкнути світлодіоди LD4, LD6. При натисканні користувальницької клавіші змінювати частоту мерехтіння світлодіода.

6.3 Робота з таймерами

Метою цього підрозділу є навчання використання таймерів для виконання певних дій у певні часові інтервали з перериванням по переповненню. Устаткування й програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite.

Основні положення щодо роботи з таймерами

Мікроконтролер STM32F407VG містить 14 таймерів. Загальний вигляд схеми керування підрахунком імпульсів наведено на рис. 6.12.

Виробник розділяє всі таймери на три типи:

1) з розширеними можливостями;

2) загального призначення;

3) базові.

Кожен таймер може мати до 4 ліній захоплення / порівняння (саме вони використовуються в режимі генерування ШІМ).

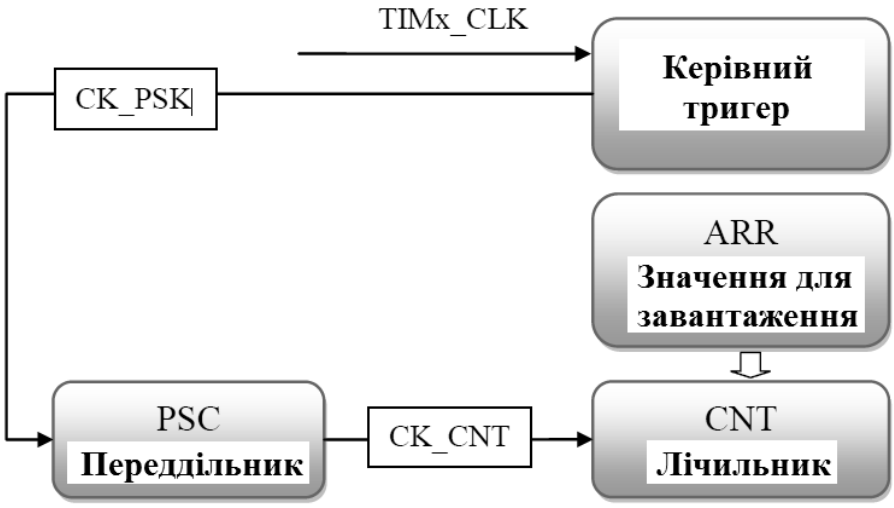


Рисунок 6.12 – Схема управління підрахунком імпульсів

Налаштування таймерів для відліку певного часу

Для того щоб таймер відраховував потрібний інтервал часу, його потрібно попередньо налаштувати. Використовуємо Clock Configuration Tool.

Для прикладу налаштуємо таймер TIM2 так, щоб він відраховував інтервал часу 1 секунду. Для початку потрібно визначити частоту шини, якій належить таймер (рис. 6.13). Переддільник (реєстр TIM_PSC) будь-якого таймера служить для налаштування кількості тактів таймера.

З рис. 6.13 видно, що частота шини таймера TIM2 становить 84 МГц.

Це означає, що за одну секунду, значення рахункового регістра таймера TIM_CNT становитиме 84 000 000, тобто таймер відрахує значення 42 000 000. Це дуже багато, тому тактами таймера можна керувати за допомогою програмованого переддільника. Його значення розраховується так:

$$\text{PSC\_VALUE} = \frac{\text{Ftim}}{\text{Fcnt}}, \tag{6.1}$$

де PSC_VALUE – значення переддільника таймера;

Ftim – вхідна частота таймера (для TIM2 становить 84 млн);

Fcnt – бажана частота рахункового регістра TIM_CNT (для TIM2 становить 8400 – 1).

Для таймера TIM2 маємо

$$PSC\_VALUE = \frac{84\,000\,000}{10\,000} = 8\,400 - 1. \tag{6.2}$$

Тепер рахунковий регістр таймера TIM2 відрахує число 10 000 за 1 секунду:

$$CNT\_VALUE = \frac{Ftim}{PSC\_VALUE} = \frac{84\,000\,000}{8\,400} = 10\,000. \tag{6.3}$$

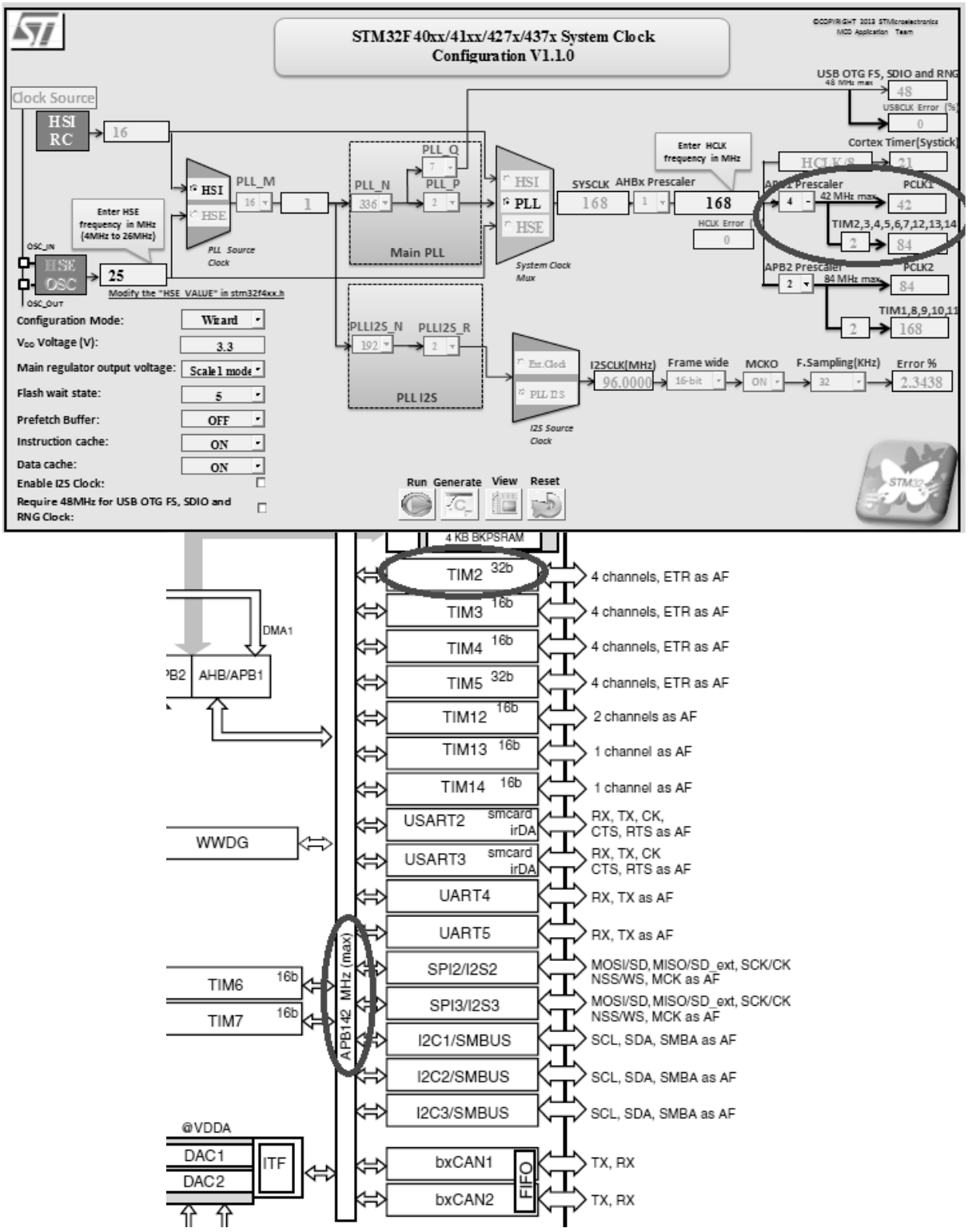


Рисунок 6.13 – Структурна схема мікроконтролера із зазначенням робочих частот усіх шин периферії

Для того щоб таймер спрацьовував через 1 секунду потрібно записати в регістр TIM_ARR (період спрацьовування таймера) таке значення

$$ARR\_VAL = T \cdot CNT\_VALUE = 1 \cdot 10\,000 = 10\,000, \qquad (6.4)$$

де T – необхідний час спрацьовування в секундах (для TIM2 – 1 секунда).

Розглянуто фрагмент програмного коду ініціалізації таймера TIM2 для того, щоб він спрацьовував через 1 секунду:

```
// Ініціалізація таймера

TIM_TimeBaseInitTypeDef tim_struct;
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2, ENABLE);
tim_struct.TIM_Prescaler = 8400-1;
tim_struct.TIM_Period = 10000;
tim_struct.TIM_ClockDivision = 0;
tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM2, & tim_struct);
// Переривання від таймера
TIM_ITConfig (TIM2, TIM_IT_Update, ENABLE);
// Увімкнення таймера
TIM_Cmd (TIM2, ENABLE);
```

При таких налаштуваннях таймера TIM2 він буде спрацьовувати через 1 секунду. Для виконання будь-яких дій (наприклад, періодичне увімкнення-вимкнення світлодіода) потрібно форматувати й дозволити переривання таймера по переповненню регістра TIM_ARR, тобто при досягненні потрібного інтервалу часу. У підпрограмі оброблювача переривань реалізується потрібна дія при спрацюванні таймера.

Приклад програми

Ця програма демонструє роботу з перериваннями і з таймерами. У ній реалізовано перемикання світлодіода LD3, який підключений до порту введення / виведення PD13, через певні інтервали часу.

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Stm32f4xx_tim.h"
#include "Misc.h"
// Ініціалізація таймера і периферії (портів зі світлодіодами)
void INTTIM_Config (void);
void GPIO_Config(Void);

int main (void)
{
GPIO_Config();
INTTIM_Config ();
while(1)
{
```

```
}
}
// Оброблювач переривання по переповненню таймера TIM2
void TIM2_IRQHandler (void) {
if (TIM_GetITStatus (TIM2, TIM_IT_Update)! = RESET) {
TIM_ClearITPendingBit (TIM2, TIM_IT_Update);
GPIOD-> ODR ^ = GPIO_Pin_13;
}
}

// Ініціалізація портів введення-виведення
void GPIO_Config(Void) {
GPIO_InitTypeDef gpio_struct;
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpio_struct.GPIO_Pin = GPIO_Pin_13;
gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
gpio_struct.GPIO_OType = GPIO_OType_PP;
gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init (GPIOD, & gpio_struct);
}

void INTTIM_Config (void)
{
// Ініціалізація переривання

NVIC_InitTypeDef nvic_struct;
nvic_struct.NVIC_IRQChannel = TIM2_IRQn;
nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
nvic_struct.NVIC_IRQChannelSubPriority = 1;
nvic_struct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init (& nvic_struct);

// Ініціалізація таймера

TIM_TimeBaseInitTypeDef tim_struct;
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2, ENABLE);
tim_struct.TIM_Prescaler = 8400-1;
tim_struct.TIM_Period = 10000;
//tim_struct.TIM_ClockDivision = 0;
tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM2, & tim_struct);
// Переривання від таймера
TIM_ITConfig (TIM2, TIM_IT_Update, ENABLE);
```

```
// Вмикаємо таймер
TIM_Cmd (TIM2, ENABLE);

}
```

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розробки і перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій, шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки і в режимі налагодження.

3. Створення нового проекту в середовищі розробки для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати і демонстрація роботи програми.

6. Аналіз отриманих результатів.

Можливі варіанти застосування таймерів

На основі наведеного прикладу можливі різні варіанти використання таймерів відповідно до табл.6.3.

Таблиця 6.3 – Варіанти використання таймерів

№ з/п	Можливе використання таймера
1	Увімкнути-вимкнути світлодіод LD4 за допомогою таймера.
2	Увімкнути-вимкнути світлодіод LD5 за допомогою таймера.
3	Увімкнути-вимкнути світлодіод LD6 за допомогою таймера.
4	Увімкнути-вимкнути світлодіод LD3 за допомогою таймера.
5	По перериванню від таймера увімкнути світлодіод LD4, вимкнути LD5, і навпаки.
6	Вмикати світлодіоди LD4, LD5 і вимикати за допомогою таймера через певний час.
7	Вмикати світлодіоди LD3, LD5 і вимикати за допомогою таймера через певний час.
8	Вмикати світлодіоди LD6, LD5 і вимикати за допомогою таймера через певний час.
9	Вмикати світлодіоди LD6, LD3 і вимикати за допомогою таймера через певний час.
10	Вмикати світлодіоди LD4, LD3 і вимикати за допомогою таймера через певний час.
11	Вмикати світлодіоди LD6, LD5, LD3, LD4 і вимикати за допомогою таймера через певний час.
12	Вмикати світлодіоди LD4, LD5 і вимикати LD3, LD6 і навпаки за допомогою таймера через певний час.

Продовження таблиці 6.3

1	2
13	Вмикати світлодіоди LD3, LD4 і вимикати LD5, LD6, і навпаки за допомогою таймера через певний час.
14	Вмикати світлодіоди LD5, LD6 і вимикати LD4, LD3, і навпаки за допомогою таймера через певний час.
15	Увімкнути-вимкнути світлодіод LD5 з інтервалом часу 1 секунда.
16	Увімкнути-вимкнути світлодіод LD6 з інтервалом часу 2 секунди.
17	Увімкнути-вимкнути світлодіод LD4 з інтервалом часу 1 секунда.
18	Увімкнути-вимкнути світлодіод LD3 з інтервалом часу 2 секунди.
19	Увімкнути-вимкнути світлодіод LD6 з інтервалом часу 4 секунди.
20	Увімкнути-вимкнути світлодіод LD3, LD6 з інтервалом часу 2 секунди.

6.4 Генерування сигналу ШІМ

Метою підрозділу є ознайомлення з можливостями генерування ШІМ за допомогою демонстраційної плати, генерування сигналу із заданими параметрами й використання його для керування зовнішніми пристроями.

Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite.

Основні відомості щодо організації роботи ШІМ

Широтно-імпульсне модулювання (ШІМ) – спосіб керування середнім значенням напруги на навантаженні шляхом зміни скважності імпульсів, керованих ключем. В основному мікроконтролери дозволяють генерувати цифровий ШІМ-сигнал різної частоти.

Оскільки виведення сигналу ШІМ не є основною функцією виводів, які підключені до світлодіодів, то необхідно виконати відповідне їх конфігурування. Для цього слід задати виконання виводами альтернативних функцій. Генерування ШІМ у них пов'язане з використанням додаткових режимів таймера.

Перед підключенням пристрою відомі такі параметри ШІМ, як частота і коефіцієнт заповнення. Для їх розрахунку в контролерах STM32 необхідно визначити значення переддільника та автоматично завантажувати значення в регістр ARR (Auto-Reload Register). Розрахунок значення, яке слід записати в переддільник, виконується так:

$$\mathrm{PSC} = \frac{\mathrm{TIMxCLK}}{\mathrm{TIMxCNT}} - 1, \qquad (6.5)$$

де PSC – значення переддільника;

TIMxCLK – вхідна частота роботи таймера;

TIMxCNT – частота лічильника.

Для отримання необхідної вихідної частоти слід записати значення в регістр ARR, яке виходить з такого співвідношення:

$$\mathrm{ARR\_VAL} = \frac{\mathrm{TIMxCNT}}{\mathrm{TIMx\_out\_freq}} - 1, \tag{6.6}$$

де ARR_VAL – значення для запису в регістр ARR;

TIMxCNT – частота лічильника;

TIMx_out_freq – потрібна вихідна частота ШІМ.

Останнім етапом є задавання потрібного коефіцієнта заповнення, що забезпечить потрібне значення напруги на виході. Це установлення проводиться за допомогою регістра захоплення / порівняння (capture / compare register, CCRx), виходячи з такого співвідношення:

$$\mathrm{D} = \frac{\mathrm{CCRx\_VAL}}{\mathrm{ARR\_VAL}} \cdot 100\%. \tag{6.7}$$

де D – коефіцієнт заповнення;

CCRx_VAL – значення в регістрі CCRx (x – номер регістра для конкретної лінії);

ARR_VAL – значення для запису в регістр ARR.

Особливістю цих мікроконтролерів є те, що в переддільник і інші регістри можна записати будь-яке значення, яке можна описати за допомогою відведеної кількості розрядів.

Видавання сигналу ШІМ на вихід не є основним режимом роботи виводів порту, а відноситься до додаткових (альтернативних) режимів. Тому попередньо потрібно задати потрібний режим в налаштуваннях порту.

Для підключення альтернативних функцій до портів введення / виводу необхідно:

1. Підключити вихід до альтернативної функції відповідного периферійного пристрою за допомогою функції GPIO_PinAFConfig.

2. Конфігурувати пін у режим виконання альтернативної функції GPIO_Mode_AF.

3. Виконати інші налаштування.

4. Викликати GPIO_Init для застосування зазначених налаштовань.

Приклад програми

Для ознайомлення з генеруванням сигналу за допомогою ШІМ наведено програму, яка послідовно плавно запалює світлодіоди LD4, LD3, LD5, LD6 на платі STM32F4Discovery циклічно [10].

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include "Stm32f4xx_tim.h"

void TimerInit () {

TIM_TimeBaseInitTypeDef time_init;
TIM_OCInitTypeDef oc_init;

// Налаштування таймера

RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM4, ENABLE);
uint16_t PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 21000000);
time_init.TIM_Period = 1000; // Період таймера
time_init.TIM_Prescaler = PrescalerValue; // Переддільник таймера
time_init.TIM_ClockDivision = 0;
time_init.TIM_CounterMode = TIM_CounterMode_Up; // Рахунок таймера по
наростанню
TIM_TimeBaseInit (TIM4, & time_init);

// Налаштування таймера для отримання ШІМ TIM_OCMode_PWM1 – прямий
ШІМ-сигнал,
// TIM_OCMode_PWM1 – інверсний,
// Прямий ШІМ-сигнал, чим більше число в CCR, тим яскравіше горять світ-
лодіоди

oc_init.TIM_OCMode = TIM_OCMode_PWM1; // Прямий ШІМ-сигнал
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0; // Значення CCR
oc_init.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC1Init (TIM4, & oc_init);
TIM_OC1PreloadConfig (TIM4, TIM_OCPreload_Enable);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
TIM_OC2Init (TIM4, & oc_init);
TIM_OC2PreloadConfig (TIM4, TIM_OCPreload_Enable);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
oc_init.TIM_Pulse = 0;
TIM_OC3Init (TIM4, & oc_init);
TIM_OC3PreloadConfig (TIM4, TIM_OCPreload_Enable);
oc_init.TIM_OutputState = TIM_OutputState_Enable;
```

```
oc_init.TIM_Pulse = 0;
TIM_OC4Init (TIM4, & oc_init);
TIM_OC4PreloadConfig (TIM4, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig (TIM4, ENABLE);

TIM_Cmd (TIM4, ENABLE);

}

void GPIOinit () {
GPIO_InitTypeDef gpio_init;

// Ініціалізація портів введення-виведення (світлодіоди)

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
gpio_init.GPIO_Mode = GPIO_Mode_AF;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init (GPIOD, & gpio_init);
GPIO_PinAFConfig (GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
GPIO_PinAFConfig (GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
GPIO_PinAFConfig (GPIOD, GPIO_PinSource14, GPIO_AF_TIM4);
GPIO_PinAFConfig (GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
}

void init () {

GPIOinit ();
TimerInit ();

}

// Затримання
void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}

int main (void)
{
```

```
init ();

int i;
while(1)
{

// Заповнення регістрів CCR для плавного увімкнення світлодіодів

for(I = 0; i <4095; i ++)
{
TIM4-> CCR1 = i;
Delay (10000);
}

for(I = 0; i <4095; i ++)
{
TIM4-> CCR2 = i;
Delay (10000);
}

for(I = 0; i <4095; i ++)
{
TIM4-> CCR3 = i;
Delay (10000);
}

for(I = 0; i <4095; i ++)
{
TIM4-> CCR4 = i;
Delay (10000);
}
// Вимкнення всіх світлодіодів і повторення всього заново
TIM4-> CCR1 = 0;
TIM4-> CCR2 = 0;
TIM4-> CCR3 = 0;
TIM4-> CCR4 = 0;
}

}
```

Як видно з прикладу, для ініціалізації таймера необхідні відразу дві структури: одна – для ініціалізації безпосередньо таймера, а інша – для задавання режиму порівняння виходу. Для отримання вихідного ШІМ-сигналу на світлодіодах використовується таймер TIM4 (див. документацію або рис. 6.14).

Рисунок 6.14 – Прив'язка каналів таймера TIM4

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення і перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки і в режимі налаштування.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати й перевірка роботи програми.

6. Аналіз отриманих результатів.

Можливі варіанти керування світлодіодами на налагоджувальній платі

На основі наведеного прикладу можливе виконання різних варіантів керування світлодіодами відповідно до табл. 6.4.

Таблиця 6.4 – Варіанти керування світлодіодами

№ з/п	Варіанти керування світлодіодами
1	2
1	Здійснити плавне увімкнення світлодіодів LD4, LD3, LD5, LD6.
2	Здійснити плавне увімкнення світлодіодів LD4, LD3.
3	Здійснити плавне увімкнення світлодіодів LD5, LD6.
4	Здійснити плавне увімкнення світлодіодів LD4, LD6.
5	Здійснити плавне увімкнення світлодіодів LD4, LD6 і їх загасання в циклі.
6	Здійснити плавне увімкнення світлодіодів LD5, LD6 і їх загасання в циклі.
7	Здійснити плавне увімкнення світлодіодів LD3, LD6 і їх загасання в циклі.
8	Здійснити плавне увімкнення світлодіодів LD4, LD6 і їх загасання в циклі.
9	Здійснити плавне увімкнення світлодіодів LD4, LD3, LD5, LD6 і їх вимкнення в циклі.
10	Плавно увімкнути й швидко вимкнути світлодіод LD4.

Продовження таблиці 6.4

1	2
11	Плавно увімкнути й швидко вимкнути світлодіод LD6.
12	Плавно увімкнути й швидко вимкнути світлодіод LD3.
13	Плавно увімкнути й швидко вимкнути світлодіод LD5.
14	Плавно увімкнути світлодіод LD4, потім швидко вимкнути його. Плавно увімкнути світлодіод LD5 і плавно його вимкнути.
15	Плавно увімкнути світлодіод LD3, потім швидко вимкнути його. Плавно увімкнути світлодіод LD4 і швидко його вимкнути.
16	Швидко увімкнути світлодіоди LD4, LD3, LD5, LD6, потім їх плавно вимкнути.
17	Швидко увімкнути світлодіоди LD4, LD3, потім їх плавно вимкнути.
18	Швидко увімкнути світлодіоди LD4, LD6, потім їх плавно вимкнути.
19	Швидко увімкнути світлодіоди LD4, LD5, LD6, потім їх плавно вимкнути.
20	Швидко увімкнути світлодіоди LD4, LD3, LD5, потім їх плавно вимкнути.

6.5 Використання USART

Метою підрозділу є навчання використанню універсального синхронного / асинхронного приймача, організація приймання й передавання даних між ПК і налагоджувальною платою STM32F4Discovery через перетворювач USB-UART. Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite, USB-UART-перетворювач на мікросхемі PL 2303 HX.

Основні положення по роботі з UART

Мікроконтролер на демонстраційній платі містить 6 приймачів-передавачів. При цьому 4 з них є синхронно-асинхронними (USART1, USART2, USART3, USART6) і 2 тільки асинхронними (UART4, UART5).

Розглянуто процес налаштування USART для приймання даних. Для цього необхідно виконати такі дії:

1. Увімкнути тактування USART і порту виходу, який використовується для роботи USART.

2. Виконати налаштування відповідних виходів, вказавши при цьому режим роботи виконання альтернативної функції.

3. Підключити виконання альтернативної функції до вказаних виходів.

4. Встановити налаштовання роботи USART (частота, розмір передачі, кількість стоп-біт, перевірка на парність).

5. Увімкнути USART.

Для передавання даних між ПК і налагоджувальною платою використовується перетворювач USB-UART. При підключенні до USB-порту комп'ютера перетворювача, після установлення необхідних драйверів, які йдуть у комплекті з USB-UART, операційна система додає віртуальний послідовний (COM) порт, який можна побачити в Диспетчері пристроїв Windows (рис. 6.15)

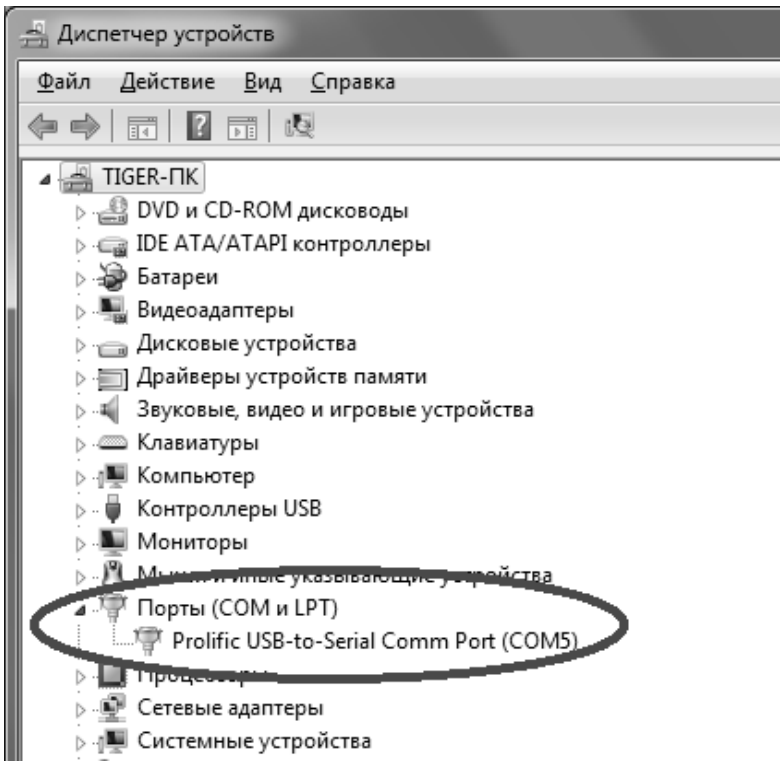


Рисунок 6.15 – Віртуальний послідовний порт

Тепер комп'ютер визначає USB-UART як послідовний порт. Усі стандартні програми для приймання й передавання даних через COM-порти (наприклад, Terminal) працюють з USB-UART, як зі звичайним послідовним портом. Отже, можна легко обмінюватися даними між мікроконтролером і ПК. Зовнішній вигляд перетворювача наведено на рис. 6.16.

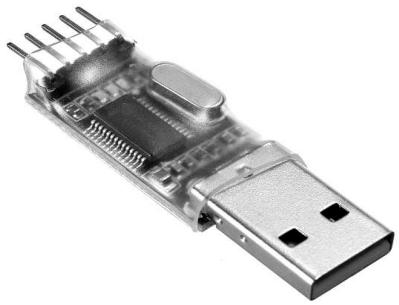


Рисунок 6.16 – Зовнішній вигляд USB-UART-перетворювача

Технічні характеристики перетворювача USB-UART:

Має на борту вбудований тактовий генератор (зовнішній резонатор не потрібний).

Максимальна швидкість передавання даних – 12 Мбит / с.

Опис виходів:

3V3 – живлення перетворювача 3,3 В;

TX – передавання даних;

RX – приймання даних;

GND – живлення перетворювача (загальний провід);

+ 5V – живлення перетворювача від 5,5 В.

Приклад програми

У цьому прикладі розглянуто приймання даних за допомогою USART3. Для передавання й приймання використовуються виходи порту C під номерами 10 (Transmit) і 11 (Receive). За командою з комп'ютера будуть запалюватися та гаснути світлодіоди LD3, LD4, LD5, LD6. У відповідь буде передаватися рядок Hello!!! з мікроконтролера на комп'ютер. Передавання й приймання даних проводиться за допомогою готового додатка Terminal та, як варіант, власного додатка, розробленого у вільному середовищі розроблення додатків SharpDevelop на мові C # платформи .NET.

Для кожного приймача-передавача виділені свої пари виводів для приймання й передавання, які можна знайти в документації до мікроконтролера (рис. 6.17).

Схему підключення перетворювача USB-UART до виводів UART3 мікроконтролера STM32F4 наведено на рис. 6.18.

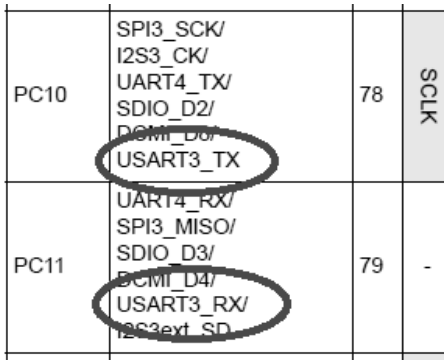


Рисунок 6.17 – Виводи UART3

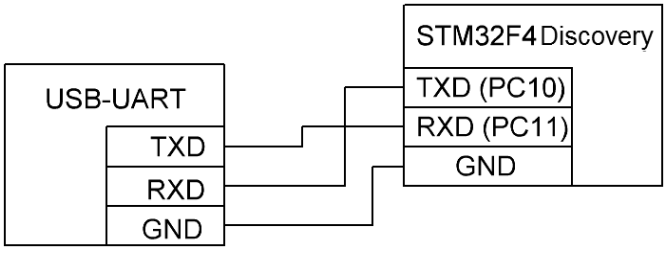


Рисунок 6.18 – Схема підключень USB-UART до мікроконтролера

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include "Stm32f4xx_usart.h"

uint8_t uart_data; // Змінна для зберігання отриманих даних з UART

// Функція відправляє байт в UART

void send_to_uart (uint8_t data) {

while(! (USART3-> SR & USART_SR_TC));

USART3-> DR = data;

}

// Функція відправляє рядок в UART

void send_str (char * string) {
uint8_t i = 0;

while(String [i]) {

send_to_uart (string [i]);
i ++;

}


send_to_uart ( '\ r');
send_to_uart ( '\ n');


}


int main (void)

{

GPIO_InitTypeDef gpioConf;

// Увімкнення портів D зі світлодіодами
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
gpioConf.GPIO_Mode = GPIO_Mode_OUT;
```

```
gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
gpioConf.GPIO_OType = GPIO_OType_PP;
gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpioConf);

// Увімкнення портів і UART3
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE);

// Порти під UART задіяні

GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3);
GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3);

// Ініціалізація портів

GPIO_InitTypeDef GPIO_InitStructure;

// Конфігуруємо UART TX UART RX як альтернативну функцію
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; // Ініціалізація входу і ви-
ходу як Alternate Function
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init (GPIOC, & GPIO_InitStructure);

// Заповнення структури налаштуваннями UARTa
USART_InitTypeDef uart_struct;

uart_struct.USART_BaudRate = 115200; // Швидкість передавання даних
(Повинна збігатися зі швидкістю в терміналі)

uart_struct.USART_WordLength = USART_WordLength_8b;
uart_struct.USART_StopBits = USART_StopBits_1;
uart_struct.USART_Parity = USART_Parity_No;
uart_struct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
uart_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

// Ініціалізація UART

USART_Init (USART3, & uart_struct);

// Увімкнення UART

USART_Cmd (USART3, ENABLE);
```

```
while (1) // Нескінченний цикл ...
{
if (USART_GetFlagStatus (USART3, USART_FLAG_RXNE)! = RESET)
{// Перевірка регістра приймання
uart_data = USART_ReceiveData (USART3); // Якщо дані є, їх потрібно отримати
// Зміна стану світлодіодів залежно від отриманих даних (символів)
// Керування роботою світлодіодів з комп'ютера
switch (uart_data)
{
case '1':
GPIOD-> ODR ^ = GPIO_Pin_12;
break;
case '2':
GPIOD-> ODR ^ = GPIO_Pin_13;
break;
case '3':
GPIOD-> ODR ^ = GPIO_Pin_14;
break;
case '4':
GPIOD-> ODR ^ = GPIO_Pin_15;
break;
case '5':
// Відсилання рядка в UART на ПК (він буде надрукований у терміналі на ПК)
send_str ( "Hello !!!");
break;
}
}
}
}
```

Приймання й передавання даних за допомогою програми Terminal наведено на рис. 6.19. Для запалювання / гасіння світлодіодів надсилаються команди 1, 2, 3, 4, відповідно. Для приймання рядка з STM32F4Discovery набирається '5' у терміналі [11].

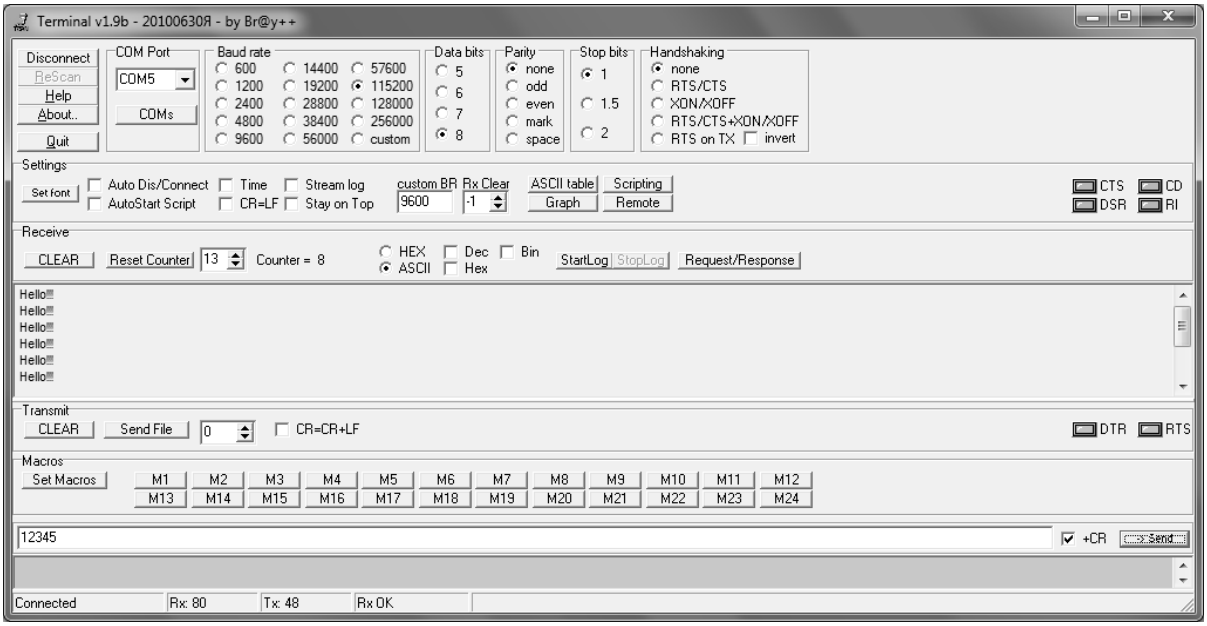


Рисунок 6.19 – Інтерфейс мікроконтролера і ПК за допомогою Terminal

Для коректного передавання даних необхідно підключити вихід перетворювача USB-UART Tx до порту мікроконтролера PC11 (Rx), а вихід Rx – до порту PC10 (Tx). Усі налаштування параметрів, такі як швидкість передавання даних, стоп-біти тощо повинні бути ідентичними в мікроконтролері й програмі, яка передає і приймає дані з послідовного порту комп'ютера (рис. 6.20) [12].

```
//Заполняем структуру настройками UARTa

USART_InitTypeDef uart_struct;

uart_struct.USART_BaudRate            = 115200;
uart_struct.USART_WordLength          = USART_WordLength_8b;
uart_struct.USART_StopBits            = USART_StopBits_1;
uart_struct.USART_Parity              = USART_Parity_No ;
uart_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
uart_struct.USART_Mode                = USART_Mode_Rx | USART_Mode_Tx;
```

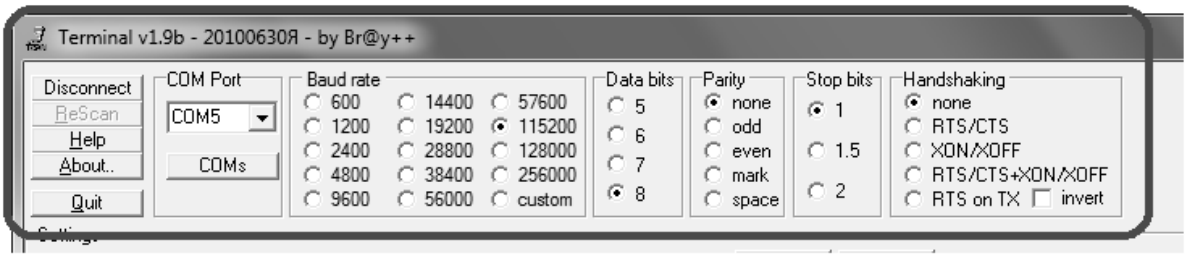


Рисунок 6.20 – Налаштування для коректного передавання даних

При розбіжності будь-яких параметрів на мікроконтролері або ПК дані можуть або взагалі не передаватися, або передаватися некоректно [13].

Для приймання й передавання даних між ПК і мікроконтролером можна використовувати власні розроблені програми.

Наведено приклад застосування додатка, розробленого в середовищі SharpDevelop на мові програмування C# (рис. 6.21) [14].

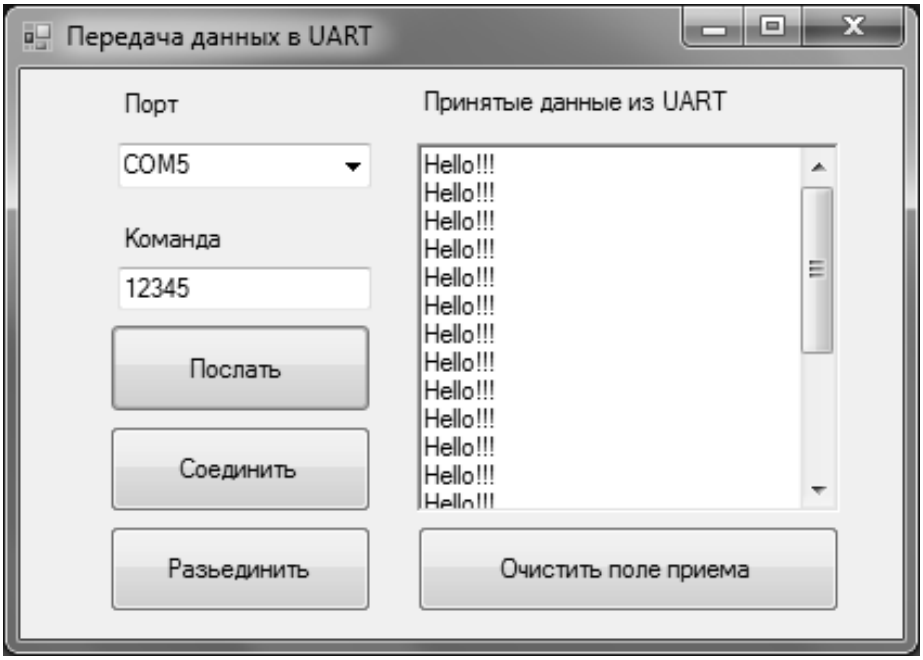


Рисунок 6.21 – Розроблений додаток на мові C# в роботі

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розробки і перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій, шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки і в режимі налагодження.

3. Створення нового проекту в середовищі розробки для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати і перевірка роботи програми.

6. Аналіз отриманих результатів

Можливі варіанти використання UART

На основі наведеного прикладу можливе виконання практичних завдань відповідно до табл. 6.5.

Таблиця 6.5 – Можливі варіанти практичного використання

№ з/п	Можливі варіанти
1	2
1	При прийманні символу «o» вмикати світлодіод LD4, при повторному прийманні цього символу вимикати світлодіод LD4.
2	При прийманні символу «d» вмикати світлодіод LD5, при прийманні символу «c» вимикати світлодіод LD5.

Продовження таблиці 6.5

1	2
3	При прийманні символу «l» вмикати світлодіод LD6, при прийманні символу «k» вимикати світлодіод LD6.
4	При прийманні символу «h» увімкнути світлодіоди LD4, LD5, LD3, LD6.
5	Відправити на ПК рядок «Hello world !!!» при прийманні символу «;».
6	При прийманні символу «a» увімкнути світлодіоди LD5, LD3, LD6.
7	При прийманні символу «h» увімкнути світлодіоди LD4, LD5.
8	При прийманні символу «j» увімкнути світлодіоди LD3, LD6.
9	При прийманні символу «t» увімкнути світлодіоди LD4, LD6.
10	Відправити на ПК рядок «Hello world !!!» при прийманні символу «y» й увімкнути світлодіод LD3.
11	Відправити на ПК рядок «ESA» при прийманні символу «z» й увімкнути світлодіод LD4.
12	Відправити на ПК рядок «ESA» при прийманні символу «k» й увімкнути світлодіод LD5.
13	Відправити на ПК рядок «Hello !!!» при прийманні символу «q» й увімкнути світлодіоди LD4, LD3.
14	При прийманні послідовності «1234» увімкнути світлодіоди LD3, LD4, LD5, LD6.
15	При прийманні послідовності «12» увімкнути світлодіоди LD3, LD4.
16	При прийманні послідовності «34» увімкнути світлодіоди LD5, LD6.
17	При прийманні послідовності «4» увімкнути світлодіоди LD3, LD6.
18	При прийманні послідовності «2» увімкнути світлодіоди LD3, LD5, LD6.
19	При прийманні символу «d» передати на ПК рядок «Hello, ESA !!! (((;;».
20	При прийманні послідовності «1234» увімкнути світлодіоди LD3, LD4, LD5, LD6 і передати рядок «ESA (((;;» на ПК.

6.6 Використання АЦП

Метою підрозділу є дослідження можливостей використання АЦП STM32F4Discovery.

Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite, змінний резистор 22 кОм.

Основні положення щодо роботи з АЦП

Мікроконтролер STM32F407VG включає в себе три АЦП. Розрядність усіх АЦП становить 12 біт. Кожен перетворювач здатний приймати сигнал з шістнадцяти зовнішніх каналів.

Крім того, до складу контролера (не налагоджувальної плати) входить датчик температури. Діапазон вхідних напруг становить 1.8...3.6 В. Датчик температури підключений до вхідного каналу ADC_IN16, який використовується для того, щоб перетворити вихідну напругу сенсора на цифрове значення.

Внутрішній датчик температури призначений для відстеження зміни температури, а не для її вимірювання, оскільки зсув показників датчика може змінюватися під час змін параметрів процесу. Тому якщо необхідно точне вимірювання абсолютних значень температури, то для цього краще використовувати зовнішній датчик [15].

Приклад програми

У цьому прикладі наведено програму, яка дозволяє в залежності від рівня вхідної напруги на вході АЦП PA1 запалювати або гасити призначені для користувача світлодіоди LD3, LD4, LD5, LD6.

Як джерело напруги буде використовуватися вихід налагоджувальної плати 3V. При обертанні ручки змінного резистора буде змінюватися рівень напруги на вході АЦП (PA1). Схему підключення змінного резистора до налагоджувальній платі STM32F4Discovery наведено на рис. 6.22.

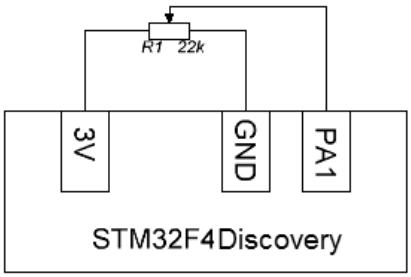


Рисунок 6.22 – Схема підключень для дослідження можливостей АЦП

Порт PA1 можна використовувати як вхід АЦП (ADC1 канал 1). Усю інформацію можна знайти в документації (рис. 6.23).

PA1	USART2_RTS/ USART4_RX/ ETH_RMII_REF_CLK/ ETH_MII_RX_CLK/ TIM5_CH2/ ADC123_IN1	24	-	-	-	-
	TIM5_CH3/					

Рисунок 6.23 – Порт АЦП

Наведено приклад програми, яка виконує необхідні дії. Вміст основного файлу main.c наведено нижче.

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include <Stm32f4xx_adc.h>

// Увімкнення й ініціалізація всіх світлодіодів

void led_init ()
{
GPIO_InitTypeDef gpioConf;

// Увімкнення порту D зі світлодіодами

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
gpioConf.GPIO_Mode = GPIO_Mode_OUT;
gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
gpioConf.GPIO_OType = GPIO_OType_PP;
gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpioConf);
}

// Ініціалізація входу АЦП - PA1

void adc_pin_init () {

GPIO_InitTypeDef gpio;

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_StructInit (& gpio);
gpio.GPIO_Mode = GPIO_Mode_AN;
gpio.GPIO_Pin = GPIO_Pin_1;
GPIO_Init (GPIOA, & gpio);
}

// Налаштування АЦП

void adc_init () {

ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef adc_init;
```

```
RCC_APB2PeriphClockCmd (RCC_APB2Periph_ADC1, ENABLE);
ADC_DeInit ();
ADC_StructInit (& ADC_InitStructure);
adc_init.ADC_Mode = ADC_Mode_Independent;
adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_CommonInit (& adc_init);
ADC_Init (ADC1, & ADC_InitStructure);
ADC_Cmd (ADC1, ENABLE);
}

// Функція читання даних з АЦП

u16 readADC1 (u8 channel) {
ADC_RegularChannelConfig (ADC1, channel, 1, ADC_SampleTime_3Cycles);
ADC_SoftwareStartConv (ADC1);
while (ADC_GetFlagStatus (ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue (ADC1);
}

int main (void) {

unsigned int bin_code;
led_init ();
adc_pin_init ();
adc_init ();

// Запуск нескінченного циклу

do {

// Читання даних з АЦП
bin_code = readADC1 (ADC_Channel_1);

// Увімкнення світлодіодів залежно від значення bin_code

if(Bin_code> = 1000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_12);
}
```

```
else
GPIO_ResetBits (GPIOD, GPIO_Pin_12);

if(Bin_code> = 2000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_13);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_13);

if(Bin_code> = 3000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_14);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_14);

if(Bin_code> = 4000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_15);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_15);

} While (1);

}
```

При збільшенні значення опору R1 по черзі спалахують чотири світлодіоди. При його зменшенні світлодіоди по черзі гаснуть.

Є кілька моментів, на які потрібно звернути особливу увагу. По-перше, при ініціалізації порту задається значення, яке потрібно для його роботи в аналоговому режимі за допомогою значення GPIO_Mode_AN. По-друге, ініціалізація й зчитування значень з АЦП. Налаштовується АЦП так, що перетворення здійснюється програмно й виконується за допомогою виклику функції readADC1.

Наведемо ще один приклад, де, крім запалювання світлодіодів на платі, буде передаватися значення коду АЦП на комп'ютер через USB-UART. Також буде виведений графік зміни кодів АЦП при зміні напруги на вході в залежності від часу (рис. 6.24).

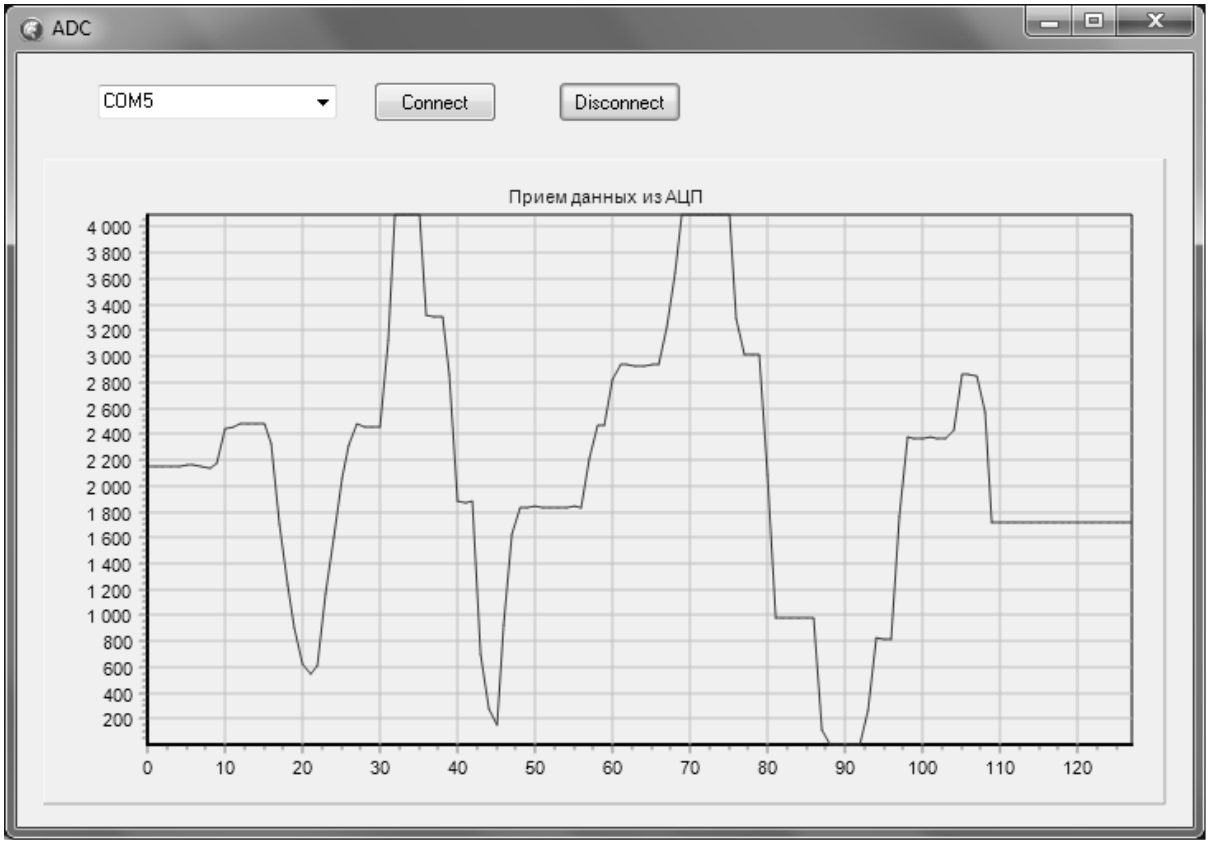


Рисунок 6.24 - Додаток для візуалізації інформації

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include "Stm32f4xx_usart.h"

// Функція відправляє байт у UART

void send_to_uart (uint8_t data) {

while(! (USART3-> SR & USART_SR_TC));

USART3-> DR = data;

}

// Функція відправляє рядок в UART

void send_str (char * string) {

uint8_t i = 0;
```

```
while(String [i]) {

send_to_uart (string [i]);

i ++;

}

send_to_uart ( '\ r');
send_to_uart ( '\ n');

}

void send_Uart (USART_TypeDef * USARTx, unsigned char c)
{
while(USART_GetFlagStatus (USARTx, USART_FLAG_TXE) == RESET) {}
USART_SendData (USARTx, c);
}

// Виведення числа в UART (максимальна довжина числа 6 цифр)
void send_int_Uart (USART_TypeDef * USARTx, long c)
{
unsigned long d = 10000000;
char temp, flag = 0;
if(C <0)
{
send_Uart (USARTx, '-');
c = -c;
}
do
{
c = c% d;
d = d / 10;
temp = c / d;
if(Temp! = 0)
{
flag = 1;
}
if(Flag == 1)
{
send_Uart (USARTx, temp + '0');
}
}
while(D> 1);
}
```

```
// Затримання
void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}

int main (void)
{

// Ініціалізація входу АЦП - PA1

void adc_pin_init () {
GPIO_InitTypeDef gpio;
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_StructInit (& gpio);
gpio.GPIO_Mode = GPIO_Mode_AN;
gpio.GPIO_Pin = GPIO_Pin_1;
GPIO_Init (GPIOA, & gpio);
}

// Налаштування АЦП
void adc_init () {

ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef adc_init;
RCC_APB2PeriphClockCmd (RCC_APB2Periph_ADC1, ENABLE);
ADC_DeInit ();
ADC_StructInit (& ADC_InitStructure);
adc_init.ADC_Mode = ADC_Mode_Independent;
adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_CommonInit (& adc_init);
ADC_Init (ADC1, & ADC_InitStructure);
ADC_Cmd (ADC1, ENABLE);
}

// Функція читання даних з АЦП

u16 readADC1 (u8 channel) {
```

```
ADC_RegularChannelConfig (ADC1, channel, 1, ADC_SampleTime_3Cycles);
ADC_SoftwareStartConv (ADC1);
while (ADC_GetFlagStatus (ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue (ADC1);
}

GPIO_InitTypeDef gpioConf;

// Увімкнення порту D зі світлодіодами

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
gpioConf.GPIO_Mode = GPIO_Mode_OUT;
gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
gpioConf.GPIO_OType = GPIO_OType_PP;
gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpioConf);

// Увімкнення порту і UART

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE);

// Порти під UART задіяні

GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3);
GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3);

// Ініціалізація портів

GPIO_InitTypeDef GPIO_InitStructure;

// Конфігурація UART TX UART RX як альтернативної функції

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

// Ініціалізація входу і виходу як Alternate Function
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; GPIO_Init (GPIOC, &
GPIO_InitStructure);

// Заповнення структури налаштуваннями UARTa

USART_InitTypeDef uart_struct;
```

```
uart_struct.USART_BaudRate = 115200; // Швидкість передавання даних (Повинна збігатися зі швидкістю в терміналі)

uart_struct.USART_WordLength = USART_WordLength_8b;

uart_struct.USART_StopBits = USART_StopBits_1;

uart_struct.USART_Parity = USART_Parity_No;

uart_struct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;

uart_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

// Ініціалізація UART

USART_Init (USART3, & uart_struct);

// Увімкнення UART

USART_Cmd (USART3, ENABLE);

unsigned int bin_code;
adc_pin_init ();
adc_init ();

while(1) // Нескінченний цикл (перевірка) ...

{

// Читання даних з АЦП

bin_code = readADC1 (ADC_Channel_1);

// Увімкнення світлодіодів залежно від значення bin_code

if(Bin_code> = 1000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_12);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_12);

if (bin_code> = 2000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_13);
}

else
```

```
GPIO_ResetBits (GPIOD, GPIO_Pin_13);

if (bin_code> = 3000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_14);
}

Else
GPIO_ResetBits (GPIOD, GPIO_Pin_14);

if(Bin_code> = 4000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_15);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_15);

Delay (1000000);

send_to_uart ( '\ r');
send_to_uart ( '\ n');
send_int_Uart (USART3, bin_code);
send_to_uart ( '\ r');
send_to_uart ( '\ n');

Delay (1000000);


}
}
```

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення й перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки й у режимі налагодження.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати й перевірка роботи програми.

6. Аналіз отриманих результатів

Можливі варіанти використання АЦП

На основі наведених прикладів у табл. 6.6 наведено варіанти використання АЦП.

Таблиця 6.6 – Можливі варіанти використання АЦП

№ з/п	Варіанти використання
1	2
1	Змінювати інтервал часу засвічування / гасіння світлодіода LD4 залежно від рівня напруги на вході АЦП PA1.
2	Змінювати інтервал часу засвічування / гасіння світлодіода LD5 залежно від рівня напруги на вході АЦП PA1.
3	Змінювати інтервал часу засвічування / гасіння світлодіода LD6 залежно від рівня напруги на вході АЦП PA1.
4	Змінювати інтервал часу засвічування / гасіння світлодіода LD3 залежно від рівня напруги на вході АЦП PA1.
5	При досягненні нижнього порогу напруги на вході АЦП PA1 засвічувати світлодіод LD3, при досягненні верхньої межі – LD4.
6	При досягненні нижнього порогу напруги на вході АЦП PA1 засвічувати світлодіод LD3, при досягненні верхньої межі – LD4.
7	При досягненні нижнього порогу напруги на вході АЦП PA1 засвічувати світлодіод LD6, при досягненні верхньої межі – LD5.
8	Передати в UART3 значення коду входу АЦП PA1. Відобразити значення в додатку Terminal.
9	При досягненні вхідної напруги на вході АЦП PA1 1,5 В засвічувати світлодіод LD4.
10	При досягненні вхідної напруги на вході АЦП PA1 1,3 В засвічувати світлодіод LD6.
11	При досягненні вхідної напруги на вході АЦП PA1 1 В засвічувати світлодіод LD5.
12	При досягненні вхідної напруги на вході АЦП PA1 2 В засвічувати світлодіод LD3. Передати значення коду АЦП на ПК.
13	Реалізувати «біжну» тінь на світлодіодах LD4, LD3. При зміненні напруги на вході АЦП змінювати швидкість «біжної» тіні.
14	Реалізувати «біжну» тінь на світлодіодах LD4, LD3, LD5. При зміненні напруги на вході АЦП змінювати швидкість «біжної» тіні.
15	Реалізувати «біжну» тінь на світлодіодах LD4, LD5. При зміненні напруги на вході АЦП змінювати швидкість «біжної» тіні.
16	Засвітити світлодіоди LD4, LD3, LD5, LD6. При досягненні 2,5 В на вході АЦП згасити світлодіоди.
17	Реалізувати «біжну» тінь на світлодіодах LD4, LD3, LD5. При зміненні напруги на вході АЦП змінювати швидкість засвічування світлодіодів.

Продовження таблиці 6.6

1	2
18	Реалізувати «біжний» вогонь на світлодіодах LD4, LD3, LD5, LD6. При зміненні напруги на вході АЦП змінювати швидкість засвічування світлодіодів.
19	Реалізувати «біжний» вогонь на світлодіодах LD4, LD5. При зміненні напруги на вході АЦП змінювати швидкість засвічування світлодіодів.
20	Реалізувати «біжний» вогонь на світлодіодах LD5, LD6. При зміненні напруги на вході АЦП змінювати швидкість засвічування світлодіодів.

6.7 Використання ЦАП

Метою підрозділу є дослідження можливостей використання ЦАП (цифроаналоговий перетворювач) STM32F4Discovery.

Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite, осцилограф RIGOL.

Основні положення щодо роботи з ЦАП

Мікроконтролер STM32F407VG включає в себе два вбудовані ЦАП. Розрядність ЦАП складає 12 біт. DAC, призначені на виходи порту A: PA4 – DAC1, PA5 – DAC2 (рис. 6.25).

PA4	SPI1_NSS/ SPI3_NSS/ USART2_CK/ DCMI_HSYNC/ OTG_HS_SOF/ I2S3_WS/ ADC12_IN4/ DAC1_OUT	29	LRCK/AIN1x	-	-
PA5	SPI1_SCK/ OTG_HS_ULPI_CK/ TIM2_CH1_ETR/ TIM8_CHIN/ ADC12_IN5/ DAC2_OUT	30	-	-	SCL/SPC

Рисунок 6.25 – Канали ЦАП

Цифроаналогові перетворювачі використовуються для конвертування цифрової інформації на сигнал аналогової форми.

Активно використовуються для відтворення звуку, формування сигналу потрібної частоти й величини напруги.

Вбудований DAC може працювати в трьох режимах:
- генерування напруги за значенням, записаним до порту даних;
- генерування пилкоподібного сигналу;
- генерування білого шуму.

Приклад програми

Як приклад розглянуто формування пилкоподібного сигналу напруги на виході АЦП. Формуватися сигнал буде з масиву цілих чисел. Оскільки розрядність ЦАП 12 біт, максимальне число, яке можна записати в регістр ЦАП, буде 4095. Це число відповідає максимальній вихідній напрузі 3 В. Шляхом нескладних арифметичних операцій можна встановити рівень напруги 0...3 В. Значення регістра ЦАП визначається так:

$$DAC_{вих} = \frac{U_{вих} \cdot 4095}{U_{\max}}, \qquad (6.8)$$

де $U_{вих}$ – необхідне значення вихідної напруги;

$U_{\max}$ – максимальне значення напруги;

$DAC_{вих}$ – значення регістра ЦАП.

У прикладі програми буде використовуватися 40 точок для формування вихідного сигналу. Заповнення регістра ЦАП буде проводитися по переповненню таймера TIM6.

Розглянуто більш детально приклад програми, яка виконує необхідні дії. Вміст основного файлу main.c наведено нижче.

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_rcc.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_dac.h"
#include "Stm32f4xx_tim.h"

// Амплітудні значення ЦАП для побудови пилкоподібної напруги
const uint16_t signal [40] = {
        0, 100, 200, 300, 400, 500, 600, 700,
        800, 900, 1000, 1100, 1200, 1300, 1400, 1500,
        1600 1700, 1800, 1900, 2100, 2200, 2300, 2400,
        2500, 2600, 2700, 2800, 2900 3000, 3100, 3200,
        3300, 3400, 3500, 3600 3700, 3800, 3900, 4000,
         };

unsigned char i = 0;

// Оброблювач переривання від таймера 6
void TIM6_DAC_IRQHandler (void) {
```

```
  TIM_ClearITPendingBit (TIM6, TIM_IT_Update); // Скидання прапорця обро-
блення переривання від таймера
  DAC_SetChannel2Data (DAC_Align_12b_R, signal [i ++]); // Запис у ЦАП чер-
гового елемента масиву й вирівнювання по правому краю
   if (i == 40) i = 0;
}

int main (void)
{
SystemInit ();
// Тактування ЦАП
RCC_APB1PeriphClockCmd (RCC_APB1Periph_DAC, ENABLE);
// Тактирование таймера TIM6
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM6, ENABLE);
// Налаштування таймера TIM6
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructInit (& TIM_TimeBaseStructure);
TIM_TimeBaseStructure.TIM_Period = 10; // Частота перебору значень масиву
TIM_TimeBaseStructure.TIM_Prescaler = SystemCoreClock / 1000;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM6, & TIM_TimeBaseStructure);
TIM_Cmd (TIM6, ENABLE);
TIM_ITConfig (TIM6, TIM_IT_Update, ENABLE); // Увімкнення переривання
по переповненню
NVIC_EnableIRQ (TIM6_DAC_IRQn); // Дозвіл TIM6_DAC_IRQn переривання

// Увімкнення DAC каналу 2
DAC_Cmd (DAC_Channel_2, ENABLE);

while(1) //
{
}
}
```

За допомогою осцилографа можна спостерігати вихідний сигнал на виході PA5 (рис. 6.26). Максимальне значення вихідного сигналу – 3 В. Період зміни вихідного сигналу залежить від налаштовань таймера.

Наступний приклад показує спільне використання ЦАП і АЦП на одному мікроконтролері. Вхід АЦП – PA1, вихід ЦАП – PA5.

Вбудований ЦАП генерує вихідний сигнал згідно з масивом даних. АЦП приймає напругу з PA5. Далі оцифрований сигнал з АЦП передається за допомогою перетворювача USB-UART на ПК. Спеціалізоване розроблене програмне забезпечення дозволяє візуалізувати дані з UART.

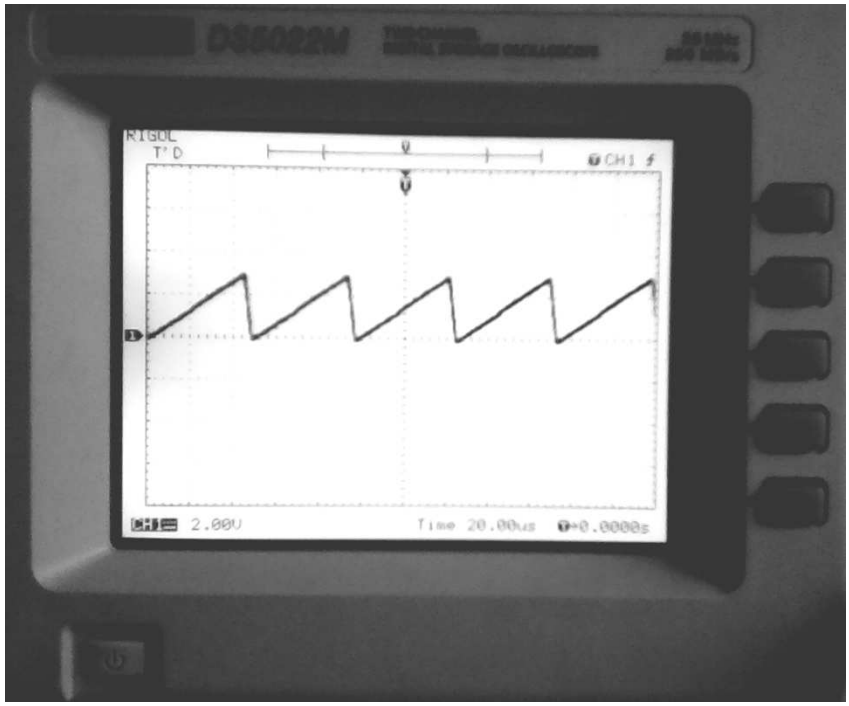


Рисунок 6.26 – Сигнал пилкоподібної напруги на виході PA5

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include "Stm32f4xx_usart.h"
#include "Stm32f4xx_dac.h"
#include "Stm32f4xx_tim.h"

// ЦАП масив //
// Амплітудні значення ЦАП для побудови пилкоподібної напруги
const uint16_t signal [40] = {
                0, 100, 200, 300, 400, 500, 600, 700,
                800, 900, 1000, 1100, 1200, 1300, 1400, 1500,
                1600 1700, 1800, 1900, 2100, 2200, 2300, 2400,
                2500, 2600, 2700, 2800, 2900 3000, 3100, 3200,
                3300, 3400, 3500, 3600    3700, 3800, 3900, 4000,
};
// Амплітудні значення ЦАП для побудови синусоїди
const uint16_t sin64 [64] = {
                659, 717, 774, 830, 883, 933, 981, 1024,
                1064, 1099, 1129, 1154, 1174, 1188, 1197, 1200,
                1197, 1188, 1 174, 1154, 1129, 1099, 1064, 1024,
                 981, 933, 883, 830, 774, 717, 659, 600,
```

```
                541, 483, 426, 370, 317, 267, 219, 176,
                136, 101, 71, 46, 26, 12, 3, 0,
                3, 12, 26, 46, 71, 101, 136, 176,
                219, 267, 317, 370, 426, 483, 541, 600};

// Параболічний закон
const uint16_t signalpar [80] = {
                1600 1521, 1 444, 1369, 1296, 1 225, 1156, 1089,
                1024, 961, 900, 841, 784, 729, 676, 625,
                576, 529, 484, 441, 400, 361, 324, 289,
                256, 225, 196, 169, 144, 121, 100, 81,
                64, 49, 36, 25, 16, 9, 4, 1, 1, 4, 9, 16, 25, 36, 49, 64,
                81, 100, 121, 144, 169, 196, 225, 256,
                289, 324, 361, 400, 441, 484, 529, 576,
                625, 676, 729, 784, 841, 900, 961, 1024,
                1089, 1156, 1 225, 1296, 1369, 1444, 1521, 1600

};
// ****************************************** //

// ЦАП переривання //
unsigned char i = 0;

// Оброблювач переривання від таймера 6
void TIM6_DAC_IRQHandler (void) {

TIM_ClearITPendingBit (TIM6, TIM_IT_Update); // Скидання прапора обробл ення переривання від таймера

// DAC_SetChannel2Data (DAC_Align_12b_R, signalpar [i ++]); // Запис в ЦАП чергового елемента масиву
 // If (i == 80) i = 0;

DAC_SetChannel2Data (DAC_Align_12b_R, 2 * sin64 [i ++]); // Запис в ЦАП чергового елемента масиву
if (I == 64) i = 0;
}
// ****************************************** //

// Функція відправляє байт в UART

void send_to_uart (uint8_t data) {

while(! (USART3-> SR & USART_SR_TC));
USART3-> DR = data;
```

```
}

// Функція відправляє рядок у UART

void send_str (char * string) {
uint8_t i = 0;

while(String [i]) {

send_to_uart (string [i]);
i ++;

}

send_to_uart ( '\ r');
send_to_uart ( '\ n');

}

// ---------------------------------------------- ----------------------

void send_Uart (USART_TypeDef * USARTx, unsigned char c)
{
while(USART_GetFlagStatus (USARTx, USART_FLAG_TXE) == RESET) {}
USART_SendData (USARTx, c);
}

// Виведення числа в UART (Максимальна довжина числа 6 цифр)
void send_int_Uart (USART_TypeDef * USARTx, long c)
{
unsigned long d = 10000000;
char temp, flag = 0;
if(C <0)
{
send_Uart (USARTx, '-');
c = -c;
}
do
{
c = c% d;
d = d / 10;
temp = c / d;
if(Temp! = 0)
{
flag = 1;
}
```

```
if(Flag == 1)
{
send_Uart (USARTx, temp + '0');
}
}
while(D> 1);
}

// Затримання
void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}

int main (void)
{
SystemInit ();

// Ініціалізація входу АЦП - PA1

void adc_pin_init () {

GPIO_InitTypeDef gpio;

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_StructInit (& gpio);
gpio.GPIO_Mode = GPIO_Mode_AN;
gpio.GPIO_Pin = GPIO_Pin_1;
GPIO_Init (GPIOA, & gpio);
}

// Налаштування АЦП

void adc_init () {

ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef adc_init;
RCC_APB2PeriphClockCmd (RCC_APB2Periph_ADC1, ENABLE);
ADC_DeInit ();
ADC_StructInit (& ADC_InitStructure);
adc_init.ADC_Mode = ADC_Mode_Independent;
adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
```

```
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_CommonInit (& adc_init);
ADC_Init (ADC1, & ADC_InitStructure);
ADC_Cmd (ADC1, ENABLE);
}

// Функція читання даних з АЦП

u16 readADC1 (u8 channel) {
ADC_RegularChannelConfig (ADC1, channel, 1, ADC_SampleTime_3Cycles);
ADC_SoftwareStartConv (ADC1);
while (ADC_GetFlagStatus (ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue (ADC1);
}

GPIO_InitTypeDef gpioConf;

// Увімкнення порту D зі світлодіодами

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpioConf.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
gpioConf.GPIO_Mode = GPIO_Mode_OUT;
gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
gpioConf.GPIO_OType = GPIO_OType_PP;
gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpioConf);

// Включення портів і UART3

RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE);

// Порти під UART задіяні

GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3);
GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3);

// Ініціалізація портів

GPIO_InitTypeDef GPIO_InitStructure;

// Конфігурування UART TX UART RX як альтернативної функції

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; // Ініціалізація входу і виходу як Alternate Function
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init (GPIOC, & GPIO_InitStructure);

// Заповнення структури налаштуваннями UARTa

USART_InitTypeDef uart_struct;

uart_struct.USART_BaudRate = 115200; // Швидкість передавання даних (Повинна збігатися зі швидкістю в терміналі)

uart_struct.USART_WordLength = USART_WordLength_8b;

uart_struct.USART_StopBits = USART_StopBits_1;

uart_struct.USART_Parity = USART_Parity_No;

uart_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;

uart_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

// Ініціалізація UART

USART_Init (USART3, & uart_struct);

// Увімкнення UART

USART_Cmd (USART3, ENABLE);

unsigned int bin_code;
adc_pin_init ();
adc_init ();

// ***************************** ЦАП *************************//

// Тактування ЦАП
RCC_APB1PeriphClockCmd (RCC_APB1Periph_DAC, ENABLE);

// Тактування таймера TIM6
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM6, ENABLE);

// Налаштування таймера TIM6
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructInit (& TIM_TimeBaseStructure);
TIM_TimeBaseStructure.TIM_Prescaler = 8400;
TIM_TimeBaseStructure.TIM_Period = 5000; // Частота перебору значень масиву
```

```
TIM_TimeBaseStructure.TIM_ClockDivision = 1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM6, & TIM_TimeBaseStructure);
TIM_Cmd (TIM6, ENABLE);

TIM_ITConfig (TIM6, TIM_IT_Update, ENABLE); // Увімкнення переривання
по переповненню
NVIC_EnableIRQ (TIM6_DAC_IRQn); // Дозвіл TIM6_DAC_IRQn переривання

// Увімкнення DAC2
DAC_Cmd (DAC_Channel_2, ENABLE);

// ********************************************** ********** //

while(1) // Нескінченний цикл ...
{

// Читання даних з АЦП

bin_code = readADC1 (ADC_Channel_1);

// Увімкнення світлодіодів залежно від значення bin_code

if(Bin_code> = 1000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_12);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_12);

if(Bin_code> = 2000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_13);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_13);

if(Bin_code> = 3000)
{
GPIO_SetBits (GPIOD, GPIO_Pin_14);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_14);

if(Bin_code> = 4000)
```

```
{
 GPIO_SetBits (GPIOD, GPIO_Pin_15);
}

else
GPIO_ResetBits (GPIOD, GPIO_Pin_15);

Delay (1000000);

// Передавання даних на ПК
send_to_uart ( '\ r');
send_to_uart ( '\ n');
send_int_Uart (USART3, bin_code);
send_to_uart ( '\ r');
send_to_uart ( '\ n');

Delay (1000000);


}
}
```

Результати спільного використання ЦАП і АЦП наведено на рис. 6.27, 6.28.

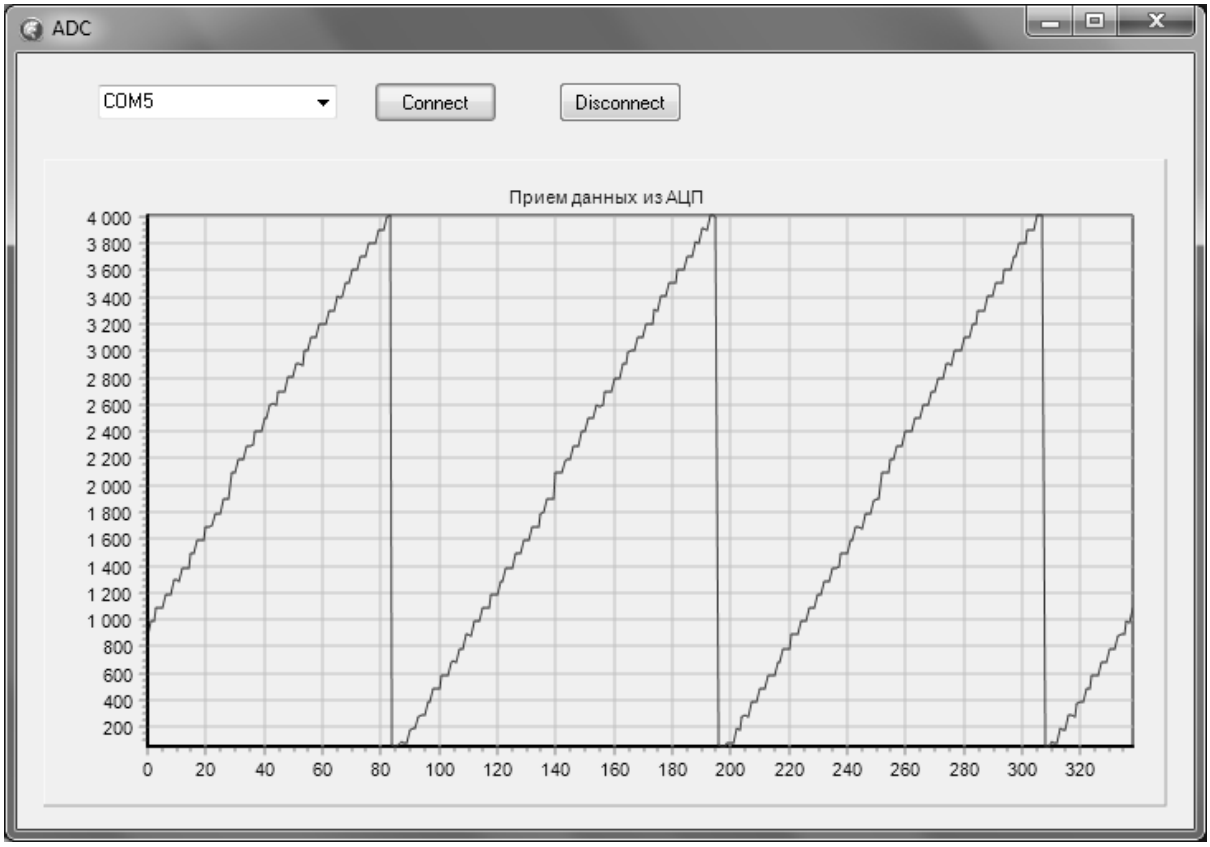


Рисунок 6.27 – Сигнал пилкоподібної напруги

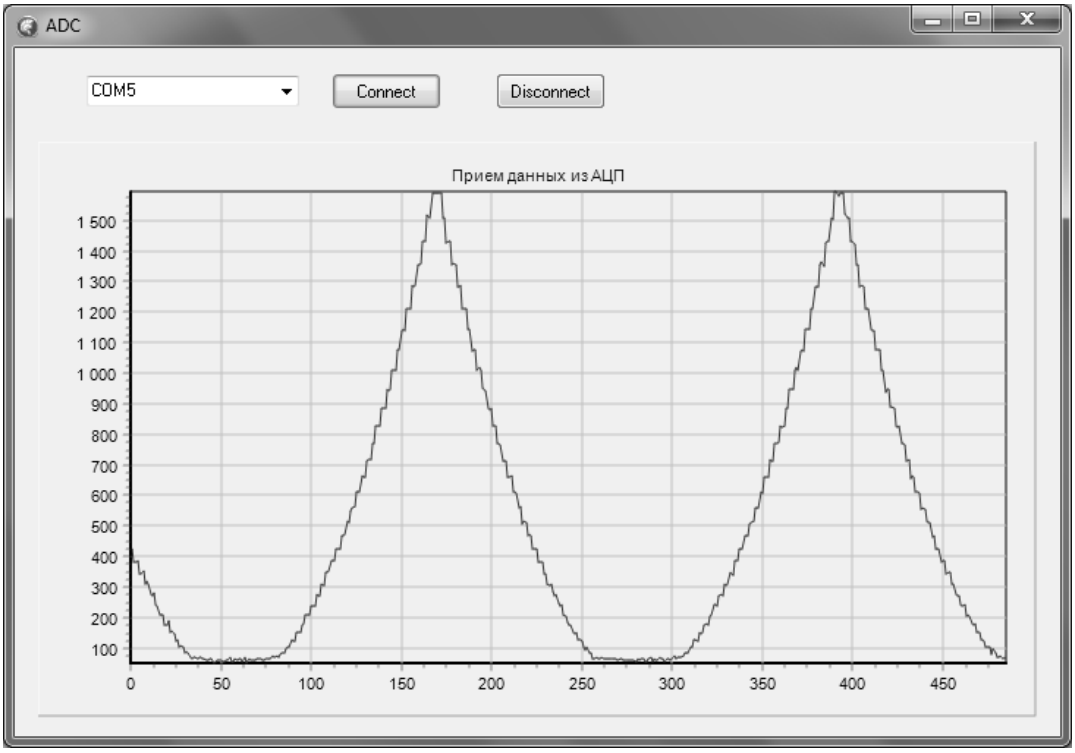


Рисунок 6.28 – Сигнал y = x2

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення і перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки й у режимі налагодження.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати йперевірка роботи програми.

6. Аналіз отриманих результатів

Можливі варіанти використання ЦАП

На основі наведених прикладів можливе використання ЦАП для виконання різних завдань (табл. 6.7).

Таблиця 6.7 – Можливі варіанти використання ЦАП

№ з/п	Можливі варіанти використання ЦАП
1	2
1	Сформувати трапецієподібний сигнал за допомогою ЦАП (амплітуда – 3 В).
2	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 1,5 В).

Продовження таблиці 6.7

1	2
3	Сформувати трапецієподібний сигнал за допомогою ЦАП (амплітуда – 2 В).
4	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 2 В, другий – 3 В).
5	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 2 В).
6	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 2 В, третій – 3 В).
7	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 0 В, другий – 1 В, третій – 2 В).
8	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 0 В, другий – 1,5 В).
9	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 1,5 В).
10	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 2,5 В).
11	Сформувати пилкоподібну напругу амплітудою 1,5 В.
12	Сформувати періодично змінюваний сигнал y = x2.
13	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 3 В).
14	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 2,5 В).
15	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 1 В).
16	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 3 В).
17	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 2 В).
18	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 1 В).
19	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 1,5 В).
20	Сформувати сигнал (перший ступінь – 2 В, другий – лінійно наростальний сигнал до 3 В).

6.8 Використання прямого доступу до пам'яті DMA (Direct memory access)

Метою підрозділу є дослідження можливостей використання DMA в STM32F4Discovery. Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite, осцилограф RIGOL.

Основні положення щодо використання DMA

DMA (Direct memory access) – прямий доступ до пам'яті, минаючи процесор. Цей механізм дозволяє виконувати будь-які дії (наприклад, передавання даних, генерування сигналу на виході ЦАП) без участі процесора.

Такий механізм істотно прискорює роботу проектів, реалізованих з використанням мікроконтролерів.

Приклад програми

Як приклад розглянуто генерування сигналу на виході ЦАП за допомогою DMA. Генерування сигналу без використання DMA застосовується рідко, тому що виконувана програма постійно переривається.

Таким чином, процесор постійно відволікається від виконання основної програми й вона працює повільніше, причому уповільнення тим більше, чим більша частота генерування переривань від таймера TIM6.

Для використання DMA потрібно розібратися з принципом генерування сигналу з використанням цієї технології.

ЦАП річ примітивна, і фактично все, на що вона чекає, це нове значення в регістрі даних для трансформування його у вихідну напругу на одному з пинів. Для організації передавання даних у модуль ЦАП введений параметр DAC_Trigger, який вказує на подію, за якою модуль DAC вимагатиме дані для себе в DMA.

Що може виступати в ролі такого тригера, можна дізнатися з керівництва по програмуванню RM00090 Reference Manual [9, с. 254] (рис. 6.29).

11.3.6 DAC trigger selection

If the TENx control bit is set, conversion can then be triggered by an external event (timer counter, external interrupt line). The TSELx[2:0] control bits determine which out of 8 possible events will trigger conversion as shown in *Table 42.*

Table 42. External triggers

Source	Type	TSEL[2:0]
Timer 6 TRGO event	Internal signal from on-chip timers	000
Timer 8 TRGO event		001
Timer 7 TRGO event		010
Timer 5 TRGO event		011
Timer 2 TRGO event		100
Timer 4 TRGO event		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

Each time a DAC interface detects a rising edge on the selected timer TRGO output, or on the selected external interrupt line 9, the last data stored into the DAC_DHRx register are transferred into the DAC_DORx register. The DAC_DORx register is updated three APB1 cycles after the trigger occurs.

If the software trigger is selected, the conversion starts once the SWTRIG bit is set. SWTRIG is reset by hardware once the DAC_DORx register has been loaded with the DAC_DHRx register contents.

Note: 1 TSELx[2:0] bit cannot be changed when the ENx bit is set.

2 When software trigger is selected, the transfer from the DAC_DHRx register to the DAC_DORx register takes only one APB1 clock cycle.

Рисунок 6.29 – Події, тригери, що викликають передавання даних із DMA в ЦАП

На рис. 6.29 зазначено перелік усіх можливих подій, і серед них вже знайомі таймери, обраний TIM6 – цей таймер у прикладі буде генерувати події, за якими DAC буде отримувати нове значення від модуля DMA й видавати на вихід (два виводи PA4 і PA5 для DAC1 і DAC2, відповідно) [16].

Потрібно налаштувати контролер прямого доступу до пам'яті (DMA) [9, с. 164] (рис. 6.30), де описані потоки й канали для DMA1 (усього в контролері 2 блоки DMA) [17].

Table 20. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S2_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Рисунок 6.30 – Канали й потоки DMA

Використовуватиметься Channel 7, потік 6 (DAC2 Channel 7 / Stream 6). Повний лістинг програми генерування сигналу на виході ЦАП з прямим доступом у пам'ять наведений далі.

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_rcc.h"
#include "Stm32f4xx_dac.h"
#include "Stm32f4xx_dma.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_tim.h"

// Амплітудні значення ЦАП для побудови синусоїди
const uint16_t sin64 [64] = {
659, 717, 774, 830, 883, 933, 981, 1024,
1064, 1099, 1129, 1154, 1174, 1187, 1197, 1200,
1197, 1188, 1174, 1154, 1129, 1099, 1064, 1024,
981, 933, 883, 830, 774, 717, 659, 600,
541, 483, 426, 370, 317, 267, 219, 176,
136, 101, 71, 46, 26, 12, 3, 0,
3, 12, 26, 46, 71, 101, 136, 176,
```

```
219, 267, 317, 370, 426, 483, 541, 600};

int main (void)
{
SystemInit (); // Налаштування тактування

// Увімкнення DMA1 GPIOA DAC TIM6
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1|
RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_DAC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM6, ENABLE);

// Налаштування вихідного піна
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_Init (GPIOA, & GPIO_InitStructure);

// Налаштування таймера TIM6
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructInit (& TIM_TimeBaseStructure);
TIM_TimeBaseStructure.TIM_Period = 10000; // Значення, що відповідає за частоту генерування
TIM_TimeBaseStructure.TIM_Prescaler = 8400;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM6, & TIM_TimeBaseStructure);
TIM_SelectOutputTrigger (TIM6, TIM_TRGOSource_Update); // Увімкнення тригера таймера
TIM_Cmd (TIM6, ENABLE);

// Налаштування DAC для видавання даних з масиву
DAC_DeInit ();
DAC_InitTypeDef DAC_InitStructure;
DAC_InitStructure.DAC_Trigger = DAC_Trigger_T6_TRGO; // Подія для отримання нового значення тригер TIM6
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable; // Посилення виходу (мінімум 0.4 вольта на виході)
DAC_Init (DAC_Channel_2, & DAC_InitStructure);

// Налаштування DMA1
// Для DAC1 потрібно використовувати зв'язки Stream5 / Channel7
// Для DAC2 використовується Stream6 / Channel7
DMA_DeInit (DMA1_Stream6);
```

```
DMA_InitTypeDef DMA_InitStructure;
DMA_InitStructure.DMA_Channel = DMA_Channel_7;
DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral;
DMA_InitStructure.DMA_BufferSize = 64; // Розмір масиву
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; // Циклічний режим
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) & DAC-> DHR12R2;
// Це регістр даних для DAC2
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;

DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t) & sin64 [0];          //
Покажчик на масив
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;

DMA_Init (DMA1_Stream6, & DMA_InitStructure); // Застосування налашто-
вань DMA

DAC_DMACmd (DAC_Channel_2, ENABLE); // Увімкнення зв'язку DAC-DMA
DAC_Cmd (DAC_Channel_2, ENABLE);        // увімкнення DAC
DMA_Cmd (DMA1_Stream6, ENABLE);                 // увімкнення DMA

// Нескінченний цикл
while(1)
{

}
}
```

Після компілювання й прошивання мікроконтролера можна спостерігати на виході PA5 сигнал за допомогою осцилографа (рис. 6.31)

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення й перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки й у режимі налагодження.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

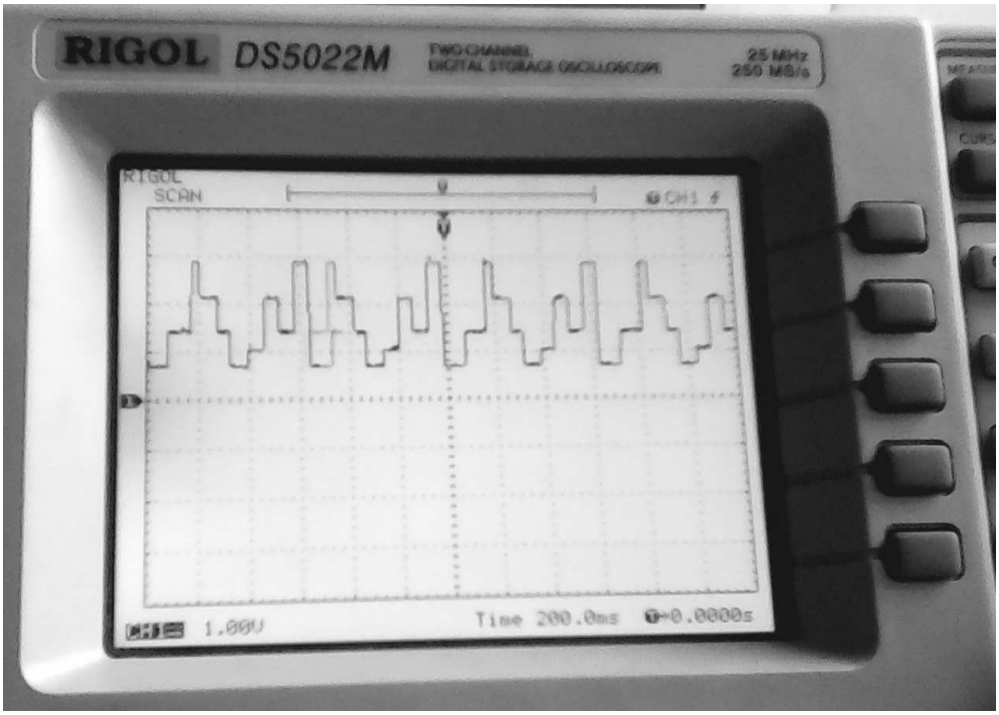


Рисунок 6.31 – Сигнал на виході ЦАП

4. Реалізація необхідної функціональності.
5. Прошивання плати й перевірка роботи програми.
6. Аналіз отриманих результатів.

Можливе застосування DMA

На основі наведених прикладів можливі й інші варіанти застосування DMA (табл. 6.8).

Таблиця 6.8 – Можливе застосування DMA

№ з/п	Варіанти застосування DMA
1	2
1	Сформувати трапецієподібний сигнал за допомогою ЦАП (амплітуда – 3 В).
2	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 1,5 В).
3	Сформувати трапецієподібний сигнал за допомогою ЦАП (амплітуда – 2 В).
4	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 2 В, другий – 3 В).
5	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, Другий – 2 В).
6	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 2 В, третій – 3 В).
7	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 0 В, другий – 1 В, третій – 2 В).

Продовження таблиці 6.8

1	2
8	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 0 В, Другий ступінь м 1,5 В).
9	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 1,5 В).
10	Сформувати ступінчастий сигнал за допомогою ЦАП (перший ступінь – 1 В, другий – 2,5 В).
11	Сформувати пилкоподібна напругу амплітудою 1,5 В.
12	Сформувати періодично змінюваний сигнал y = x2.
13	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 3 В).
14	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 2,5 В).
15	Сформувати трикутний сигнал за допомогою ЦАП (амплітуда – 1 В).
16	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 3 В).
17	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 2 В).
18	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 1 В).
19	Сформувати сигнал (перший ступінь – 0 В, другий – лінійно наростальний сигнал до 1,5 В).
20	Сформувати сигнал (перший ступінь – 2 В, другий – лінійно наростальний сигнал до 3 В).

6.9 Вимірювання періоду імпульсного сигналу

Метою підрозділу є вимірювання періоду імпульсного сигналу з використанням одного з каналів таймера, налаштованих на режим захоплення.

Обладнання та програмне забезпечення: плата STM32F4Discovery, середовище розроблення Atollic TrueSTUDIO® for ARM® Version: 4.3.1 Lite, USB-UART перетворювач на мікросхемі PL 2303 HX, осцилограф RIGOL.

Короткі теоретичні відомості

Канали таймерів мікроконтролера STM32F4Discovery можуть використовуватися для захоплення сигналу. Захоплення може відбуватися по зростальному фронту, по спадному та по обох фронтах. Усе це програмно налаштовується при розробленні програми для прошивання в мікроконтролер.

При настанні події захоплення сигналу вміст рахункового регістра таймера переноситься в його регістр захоплення-порівняння. Далі вміст регістра захоплення-порівняння використовуваного каналу таймера необхідно

перенести в будь-яку змінну. Таким чином, при кожній події захоплення сигналу в певних змінних зберігається значення кількості тактів таймера.

Шляхом вирахування кількості тактів таймера поточного захоплення й попереднього (захоплення повинне в такому разі бути налаштоване по наростальним або спадним фронтах) можна отримати значення періоду сигналу в кількості тактових імпульсів. Знаючи тривалість одного такту, нескладно визначити весь період сигналу:

$$T_{\text{сигналу}} = N_{\text{тактів}} \cdot \Delta t_{\text{т}} \tag{6.9}$$

де $T_{\text{сигналу}}$ – період сигналу;

$N_{\text{тактів}}$ – період у тактових імпульсах;

$\Delta t_{\text{т}}$ – тривалість одного такту таймера в секундах:

$$\Delta t_{\text{т}} = \frac{T}{ARR\_VAL}, \tag{6.10}$$

де T – період таймера в секундах;

ARR_VAL – значення регістра таймера ARR.

Графічну ілюстрацію захоплення сигналу по зростальними фронтами наведено на рис. 6.32.

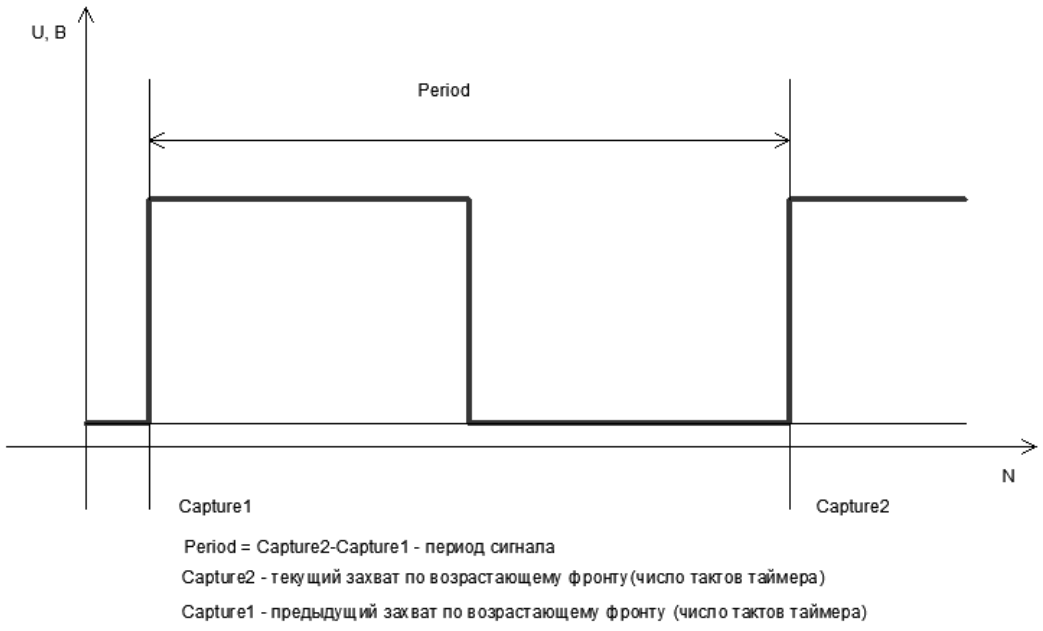


Рисунок 6.32 – Захоплення сигналу й визначення його періоду

Приклад програми

Як приклад розглянуто захоплення сигналу по передньому фронту. Таким чином, можна виміряти період і частоту вхідного сигналу. Для здійснення захоплення використовується другий канал таймера TIM3, який використовує вивід мікроконтролера PC7 згідно з документацією (рис. 6.33).

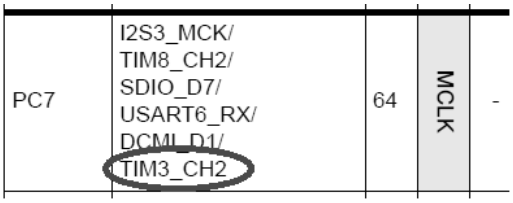


Рисунок 6.33 – Другий канал таймера TIM3 в прив'язці до виходу порту 3 PC7

Вихідний сигнал, період якого необхідно виміряти, генерується на виході порту D PD13. Таким чином, мікроконтролер STM32F4 виробляє генерування сигналу і одночасно його вимір. Для здійснення вимірювань необхідно замкнути виводи PC7 і PD13 для того, щоб мікроконтролер реєстрував зростаючі фронти згенерованого сигналу для вимірювання його періоду [18]. Кількість захоплених тактів таймера TIM3 передається на персональний комп'ютер за допомогою перетворювача USB-UART. Для прийому даних від мікроконтролера на стороні персонального комп'ютера розроблено спеціальний додаток на мові C#. Схема підключень використовуваного обладнання приведена на рис. 6.34.

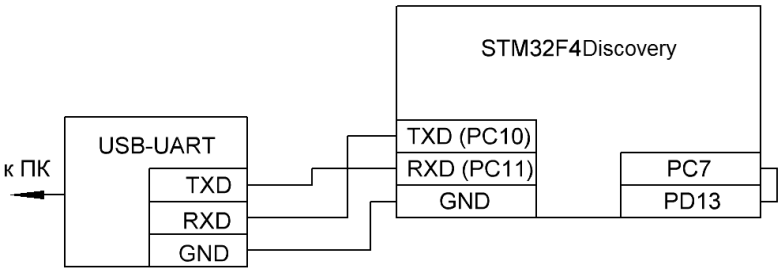


Рисунок 6.34 – Схема підключень обладнання

Період і частота вимірюваного сигналу спочатку були зафіксовані за допомогою осцилографа RIGOL (рic. 6.35)

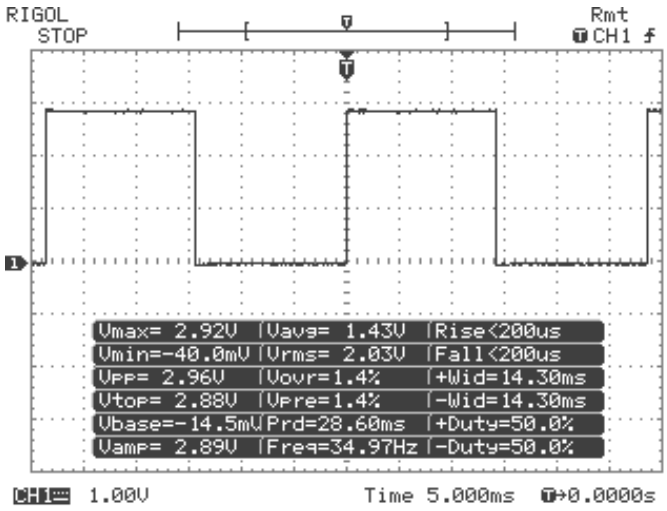


Рисунок 6.35 – Осцилограма сигналу на виході порту мікроконтролера PD13

З рис. 6.35 видно, що період сигналу становить 28,60 мс і відповідно його частота – 34,97 кГц.

Лістинг демонстраційної програми для мікроконтролера STM32F4Discovery для демонстрації можливостей вимірювання періоду сигналу за допомогою захоплення сигналу на другому каналі таймера TIM3 наведено нижче. Захоплена кількість тактів таймера відповідає періоду сигналу. Захоплення сигналу й розрахунок його періоду відбувається по перериванню таймера при захопленні на другому каналі (CC2) [19].

Лістинг програми

```
#include "Stm32f4xx.h"
#include "Stm32f4xx_gpio.h"
#include "Stm32f4xx_rcc.h"
#include "Misc.h"
#include "Stm32f4xx_tim.h"
#include "Stm32f4xx_conf.h"
#include "Stm32f4xx_it.h"

// Перемикання світлодіода

int i = 1;

void TimerInit (void);
void GPIOinit (void);

volatile uint16_t capture1 = 0, capture2 = 0;
volatile uint8_t capture_is_ready = 0, capture_ignore = 0;
uint32_t period = 0xFFFF;

uint32_t uart_data = 0;

// Затримання
void Delay (uint32_t nCount)
{
while(NCount--)
{
}
}

// Функція відправляє байт в UART

void send_to_uart (uint8_t data) {

while(! (USART3-> SR & USART_SR_TC));

USART3-> DR = data;
```

```
}

// Функція посилає біт в UART
void send_Uart (USART_TypeDef * USARTx, unsigned char c)
{
while(USART_GetFlagStatus (USARTx, USART_FLAG_TXE) == RESET) {}
USART_SendData (USARTx, c);
}

// Функція виводить число в UART. Максимальна довжина числа 6 цифр
void send_int_Uart (USART_TypeDef * USARTx, long c)
{
unsigned long d = 10000000;
char temp, flag = 0;
if(C <0)
{
send_Uart (USARTx, '-');
c = -c;
}
do
{
c = c% d;
d = d / 10;
temp = c / d;
if(Temp! = 0)
{
flag = 1;
}
if(Flag == 1)
{
send_Uart (USARTx, temp + '0');
}
}
while(D> 1);
}

// Тіло основної програми
int main (void)

// Ініціалізація периферійних пристроїв і таймера

GPIOinit ();
TimerInit ();

while(1)
```

```
        {

// Формування сигналу на виході PD13
Delay (300000);
GPIO_SetBits (GPIOD, GPIO_Pin_13);
Delay (300000);
GPIO_ResetBits (GPIOD, GPIO_Pin_13);
if(USART_GetFlagStatus (USART3, USART_FLAG_RXNE)! = RESET)
{// Перевіряємо, чи не пустий регістр приймання

// Отримання даних з UART
uart_data = USART_ReceiveData (USART3);

// Передавання обчисленого періоду на ПК
if(Uart_data == '1') {

NVIC_DisableIRQ (TIM3_IRQn);
send_int_Uart (USART3, period);
send_to_uart ( '\ r');
send_to_uart ( '\ n'); }
NVIC_EnableIRQ (TIM3_IRQn);
}
}
}

// Оброблювач преривання по переповненню таймера TIM3
void TIM3_IRQHandler (void) {
// Переривання по переповненню таймера
if (TIM_GetITStatus (TIM3, TIM_IT_Update)! = RESET) {

TIM_ClearITPendingBit (TIM3, TIM_IT_Update);

// Пропускаємо перші два виміри
capture_is_ready = 0;
capture1 = 0;
capture2 = 0;
capture_ignore = 0;

// Світлосигналізація кожну секунду для індикації переповнення таймера
if(I == 1) {
GPIO_SetBits (GPIOD, GPIO_Pin_14);
i = 0;
}
else
{
GPIO_ResetBits (GPIOD, GPIO_Pin_14);
```

```
i = 1;
}
}

// Переривання по захопленню сигналу
if (TIM_GetITStatus (TIM3, TIM_IT_CC2)! = RESET) {
TIM_ClearITPendingBit (TIM3, TIM_IT_CC2);

// Спочатку ігноруємо два виміри (інакше вимірювання можуть бути неко-
ректними)

if(Capture_ignore == 2)
{
capture_is_ready = 1;
}

if(Capture_ignore == 1)
{
capture2 = TIM_GetCapture2 (TIM3);
}

// Можна вимірювати
if(Capture_is_ready == 1)
{
NVIC_DisableIRQ (TIM3_IRQn);

// Запам'ятовування попереднього вимірювання і зчитування поточного
capture1 = capture2;
capture2 = TIM_GetCapture2 (TIM3);

// Обчислення періоду
if(Capture1> capture2)
{
period = capture1 - capture2;
}
else
{
period = capture2 - capture1;
}
NVIC_EnableIRQ (TIM3_IRQn);
}
else
{
capture_ignore = capture_ignore + 1;
}
}
```

```
// Оброблення переривання Over capture (За необхідності)
if (TIM_GetFlagStatus (TIM3, TIM_FLAG_CC2OF)! = RESET)
{
TIM_ClearFlag (TIM3, TIM_FLAG_CC2OF);
// ...
}
}

// Підпрограма ініціалізації таймера
void TimerInit (void) {

TIM_TimeBaseInitTypeDef time_init;

// Ініціалізація переривання
NVIC_InitTypeDef nvic_struct;
nvic_struct.NVIC_IRQChannel = TIM3_IRQn;
nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
nvic_struct.NVIC_IRQChannelSubPriority = 1;
nvic_struct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init (& nvic_struct);

// Налаштування таймера (період 1 сек)
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM3, ENABLE);
uint32_t PrescalerValue = 8400-1;
time_init.TIM_Period = 10000-1;
time_init.TIM_Prescaler = PrescalerValue;
time_init.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit (TIM3, & time_init);

// Встановлення каналів захоплення
TIM_ICInitTypeDef timer_ic;

// Другий канал захоплення
timer_ic.TIM_Channel = TIM_Channel_2;

// Захоплення по наростальних фронтах
timer_ic.TIM_ICPolarity = TIM_ICPolarity_Rising;

// Захоплення безпосередньо з виведення
timer_ic.TIM_ICSelection = TIM_ICSelection_DirectTI;

// Захоплення по кожному наростанні фронтів
timer_ic.TIM_ICPrescaler = TIM_ICPSC_DIV1;

// Фільтр вимкнений (корисний для усунення брязкоту)
timer_ic.TIM_ICFilter = 0;
```

```
// Ініціалізація структури
TIM_ICInit (TIM3, & timer_ic);

// Дозвіл переривання по захопленню на другому каналі та переповненню таймера TIM3
TIM_ITConfig (TIM3, TIM_IT_CC2, ENABLE);
TIM_ITConfig (TIM3, TIM_IT_Update, ENABLE);

// Увімкнення таймера
TIM_Cmd (TIM3, ENABLE);
}

// Ініціалізація портів введення-виведення таймера і UART
void GPIOinit (void) {
GPIO_InitTypeDef gpio_init;
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
gpio_init.GPIO_Pin = GPIO_Pin_7;
gpio_init.GPIO_Mode = GPIO_Mode_AF;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOC, & gpio_init);

// Використання виведення порту PC7 як вхід таймера для захоплення
GPIO_PinAFConfig (GPIOC, GPIO_PinSource7, GPIO_AF_TIM3);

// Налаштування порту D PD13 для виведення сигналу, період якого необ-хідно // виміряти
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
gpio_init.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14;
gpio_init.GPIO_Mode = GPIO_Mode_OUT;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
gpio_init.GPIO_OType = GPIO_OType_PP;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init (GPIOD, & gpio_init);

// Увімкнення UART3
RCC_APB1PeriphClockCmd (RCC_APB1Periph_USART3, ENABLE);

// Залучення портів під UART3
GPIO_PinAFConfig (GPIOC, GPIO_PinSource10, GPIO_AF_USART3);
GPIO_PinAFConfig (GPIOC, GPIO_PinSource11, GPIO_AF_USART3);
// Ініціалізація портів
GPIO_InitTypeDef GPIO_InitStructure;
```

```
// Конфігурування UART TX UART RX як альтернативної функції
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

// Ініціалізація виходу порту як альтернативної функції
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init (GPIOC, & GPIO_InitStructure);

// Заповнення структури налаштуваннями UARTa
USART_InitTypeDef uart_struct;

// Швидкість передавання даних (Повинна збігатися зі швидкістю в терміналі)
uart_struct.USART_BaudRate = 115200;
uart_struct.USART_WordLength = USART_WordLength_8b;
uart_struct.USART_StopBits = USART_StopBits_1;
uart_struct.USART_Parity = USART_Parity_No;
uart_struct.USART_HardwareFlowControl                                    =
USART_HardwareFlowControl_None;
uart_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

// Ініціалізація UART
USART_Init (USART3, & uart_struct);

// Увімкнення UART
USART_Cmd (USART3, ENABLE);
}
```

Після прошивання й здійснення всіх необхідних підключень мікроконтролер генерує вихідний сигнал, і він також може здійснити вимірювання періоду цього сигналу при виклику обробника переривань по захопленню сигналів при наростанні фронту імпульсів.

Результат вимірювання наведено на рис. 6.36 у вигляді інтерфейсу, розробленого на боці персонального комп'ютера у вигляді додатка з масивом чисел тактів, захоплених таймером.

Використовуючи вирази (6.9), (6.10), можна отримати значення періоду сигналу в секундах. Тривалість одного такту при періоді таймера TIM3 становить 1/10 000 = 0,0001 с. Період сигналу становить 286 · 0,0001 = 0,0286 с або 28,6 мс. Таким чином, значення періоду імпульсу при вимірюванні за допомогою осцилографа збіглося з виміряним за допомогою мікроконтролера, що доводить правильність роботи програм мікроконтролера для проведення вимірювання імпульсів.

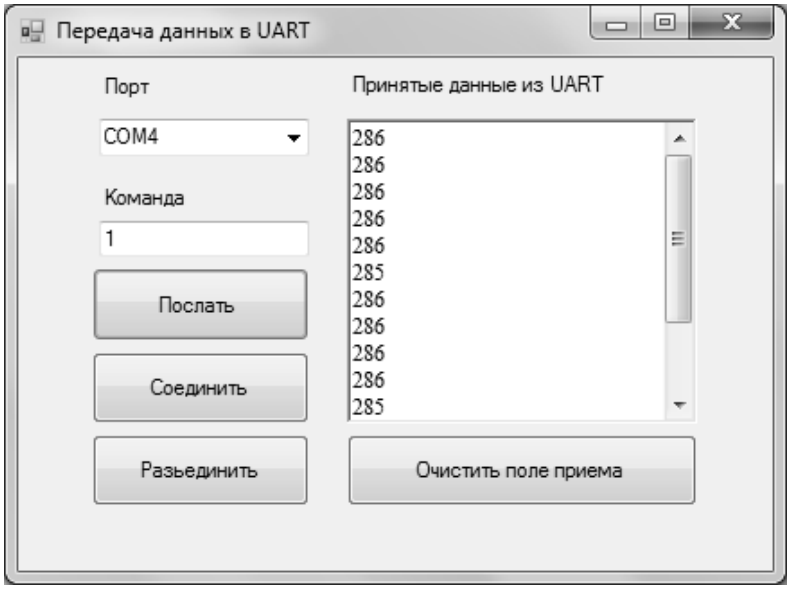


Рисунок 6.36 – Додаток для приймання значень періоду вимірюваного сигналу з мікроконтролера

Порядок роботи

1. На основі коду прикладу програми створення свого проекту в середовищі розроблення й перевірка його працездатності.

2. Ознайомлення з роботою використовуваних функцій шляхом перегляду вихідного коду, а також коментарів у вихідних файлах бібліотеки й у режимі налагодження.

3. Створення нового проекту в середовищі розроблення для виконання індивідуального завдання.

4. Реалізація необхідної функціональності.

5. Прошивання плати й перевірка роботи програми.

6. Аналіз отриманих результатів

Можливі варіанти застосування режиму захоплення-порівняння

Для вимірювання сигналу використовувати канали таймера TIM2. Налаштувати таймер і його канали для захоплення задніх фронтів сигналу.

РОЗДІЛ 7
ЗАСТОСУВАННЯ MATLAB SIMULINK ДЛЯ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ STM32F4

В останні роки в університетських та інженерно-технічних колах світу спостерігається інтенсивне поширення нової комп'ютерної системи здійснення математичних розрахунків – системи Matlab.

Головні переваги «мови технічних обчислень» Matlab, які вигідно відрізняють її серед інших існуючих нині математичних систем і пакетів, полягають у такому [20]:

1. Система Matlab спеціально створена для проведення саме інженерних розрахунків: математичний апарат, який використовується в ній, гранично наближений до сучасного математичного апарату інженера й вченого і спирається на обчислення з матрицями, векторами й комплексними числами; графічне подання функціональних залежностей тут організовано у формі, яку вимагає саме інженерна документація.

2. Мова програмування системи Matlab дуже проста, близька до мови BASIC, посильна будь-якому початківцю. Вона містить усього кілька десятків операторів. Незначна кількість операторів тут компенсується великою кількістю процедур і функцій, зміст яких легко зрозумілий користувачеві з відповідною математичною й інженерною підготовкою.

3. На відміну від більшості математичних систем, Matlab є відкритою системою. Це означає, що практично всі процедури й функції Matlab доступні не тільки для використання, але й для коригування та модифікування. Matlab – система, яка може розширюватися користувачем за його бажанням створеними ним програмами й процедурами (підпрограмами). Її легко пристосувати до вирішення задач певних класів.

4. Дуже зручною є можливість використовувати практично всі обчислювальні можливості системи в режимі надзвичайно потужного наукового калькулятора. У той же час можна складати власні окремі програми з метою багаторазового їх використання для досліджень. Це робить Matlab незамінним засобом проведення наукових розрахункових досліджень [21].

5. Останні версії Matlab дозволяють легко інтегрувати її з текстовим редактором Word, що робить можливим використання при створенні текстових документів обчислювальних і графічних можливостей Matlab.

Нові версії Matlab дозволяють за допомогою спеціальних бібліотек програмувати різні мікроконтролери, такі як Texas Instrument, STM32, Arduino. Ці бібліотеки необхідно коректно встановити в середовищі Matlab Simulink. Далі їх можна знайти в загальній бібліотеці Matlab Simulink. Використання спеціальних бібліотек для програмування мікроконтролера дозволяє значно полегшити процес програмування. Тобто програмування стає наочним і візуальним, що дуже зручно для інженера.

Після компіляції зібраної моделі генерується програмний код на мові програмування С. За необхідності програмний код можна модифікувати. Також створюється бінарний файл створеної програми, який згодом завантажується в мікроконтролер. Таким чином можна створювати програми для мікроконтролера за допомогою створення візуальної математичної моделі. Знання складних мов програмування, таких як С, при програмуванні за допомогою середовища моделювання Matlab не потрібно. Тому програмування мікроконтролерів за допомогою Matlab Simulink є актуальним.

Алгоритм налаштування Matlab Simulink для програмування мікроконтролера STM32F4Discovery

Загальний алгоритм налаштування середовища моделювання Matlab Simulink наведено на рис. 7.1.

Рисунок 7.1 – Алгоритм налаштування Matlab Simulink для програмування мікроконтролера STM32F4discovery

Для установлення необхідних бібліотек-доповнень для здійснення програмування мікроконтролера STM32F4Discovery необхідно на панелі інструментів у меню Add-Ons вибрати пункт Get Hardware Support Packages (рис. 7.2).

Для програмування мікроконтролера STM32F4Discovery зі списку пропонованих пакетів підтримання необхідно вибрати пункт STMicroelectronics STM32F4Discovery (рис. 7.3).

У вікні можна вибрати установлення пакетів підтримання з Інтернету, папки розташування файлів бібліотек, завантаження або видалення пакетів підтримання. У цьому випадку вибрано Install from Internet (встановити з Інтернету).

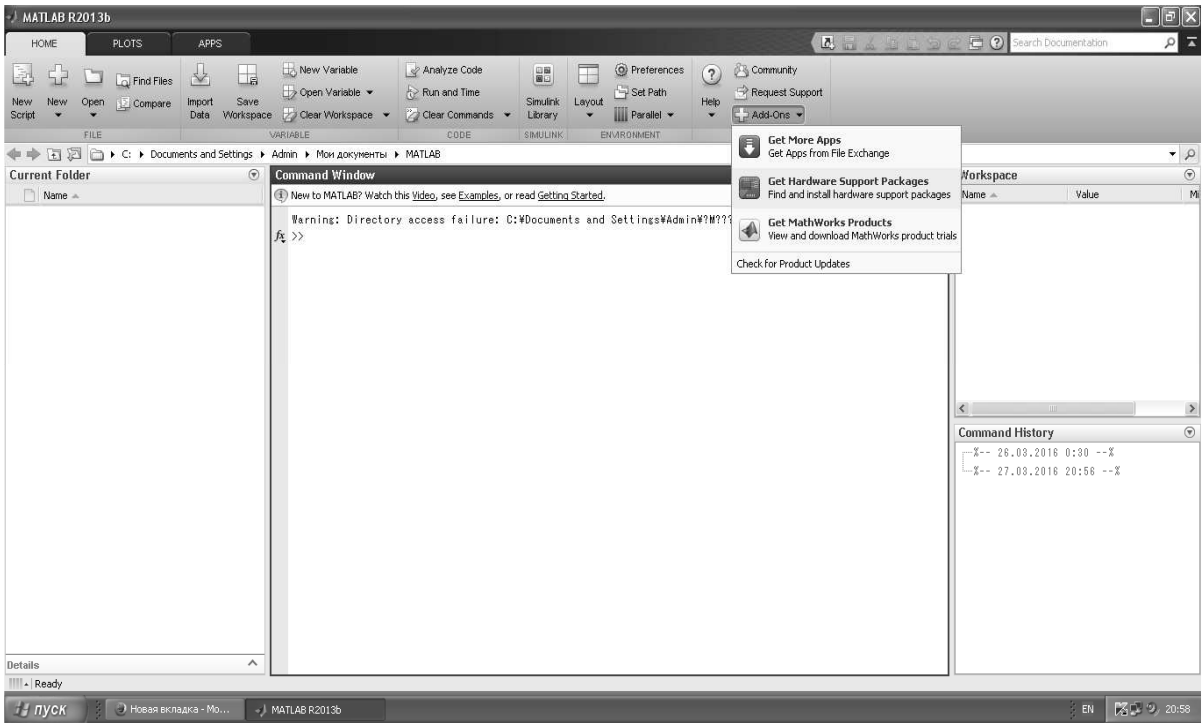


Рисунок 7.2 – Початок встановлення необхідних бібліотек

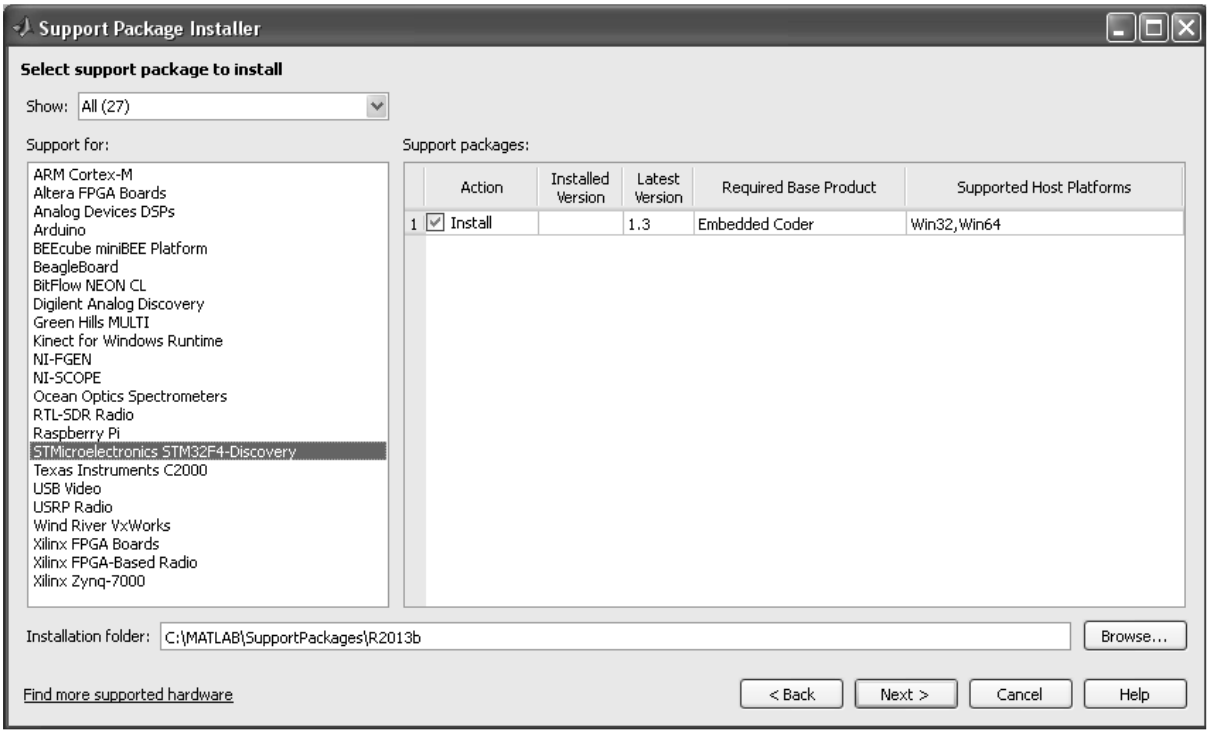


Рисунок 7.3 – Вибір пакетів підтримання для мікроконтролера STM32F4Discovery

Для того щоб продовжити установлення пакетів підтримання, необхідно натиснути Log In та у вікні ввести власний email і пароль від облікового запису, зареєстрованого на сайті www.mathworks.com.

У разі якщо відсутній обліковий запис, його можна створити, натиснувши Create an account в наведеному на цьому слайді вікні або на сайті www.mathworks.com. Після створення облікового запису необхідно ввести власний email і пароль і натиснути Log In. Після чого користувач отримує можливість ознайомитися з умовами ліцензійної угоди допоміжного програмного забезпечення від MathWorks.

Далі користувачеві буде подано список встановлюваного програмного забезпечення сторонніх виробників і звідки воно буде завантажено та посилання на ліцензійні угоди програмного забезпечення. При продовженні установлення відбувається прийняття ліцензійних угод автоматично.

У наступному вікні буде вказано, які пакети підтримання потрібно встановити і шлях їх установлення. Необхідно підтвердити установлення, натиснувши клавішу Install.

Після цього відкриється вікно, у якому буде відображатися процес завантаження програмного забезпечення. Необхідно дочекатися завершення завантаження й установлення.

Далі з'явиться вікно, яке підтверджує успішне завершення першого етапу установлення пакетів підтримання й пропонує продовжити установлення.

У наступному вікні буде запропоновано вибрати встановлюваний пакет підтримання STMicroelectronics STM32F4Discovery.

Після буде наведено список необхідного додаткового програмного забезпечення. При натисканні клавіші download відкриється сайт, з якого можна завантажити необхідне програмне забезпечення. Для можливості його завантаження необхідна попередня реєстрація на цьому сайті.

Після завантаження необхідного програмного забезпечення в наступному вікні необхідно буде вказати шлях до скачаних файлів і підтвердити установлення, натиснувши клавішу Validate.

Потім буде запропоновано перевірити правильність встановлення програмного забезпечення, його версії та шляхи до скачаних файлів. Розташування блоку Target Setup наведено на рис. 7.4.

Налаштування програматора Target Setup для створення програми для мікроконтролера STM32F4Discovery наведено на рис. 7.5,7.6.

Розташування блоку цифро-аналогового перетворювача в бібліотеці компонентів Matlab Simulink Simulink Library Browser наведено на рис. 7.7.

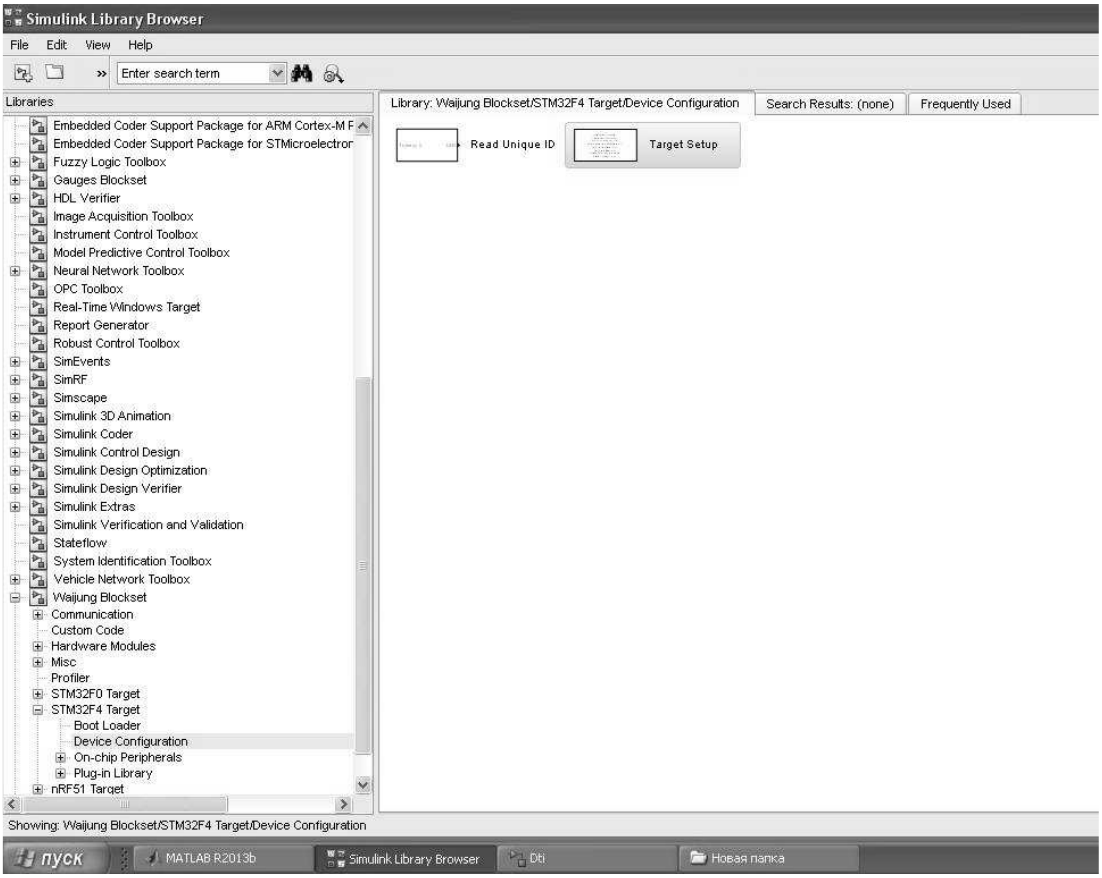


Рисунок 7.4 – Розташування блоку Target Setup

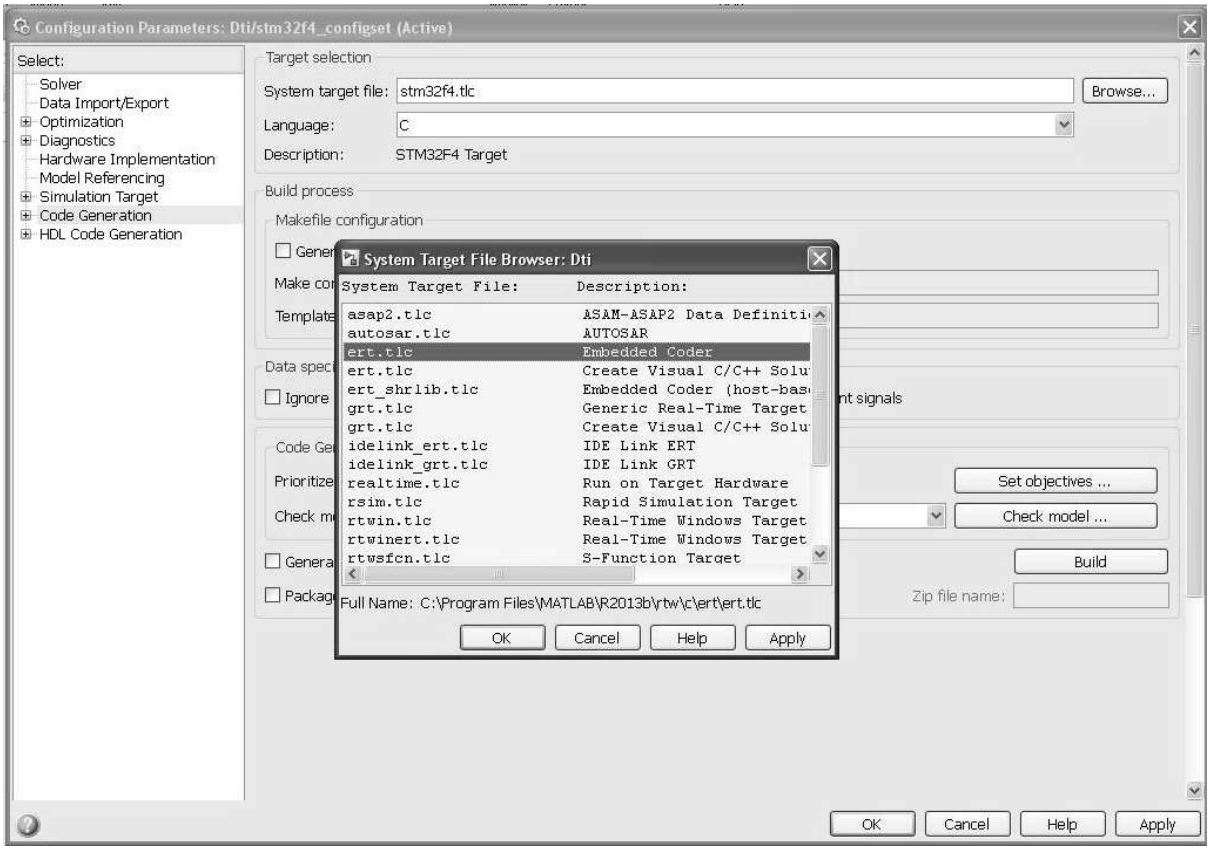


Рисунок 7.5 – Налаштування блоку Target Setup

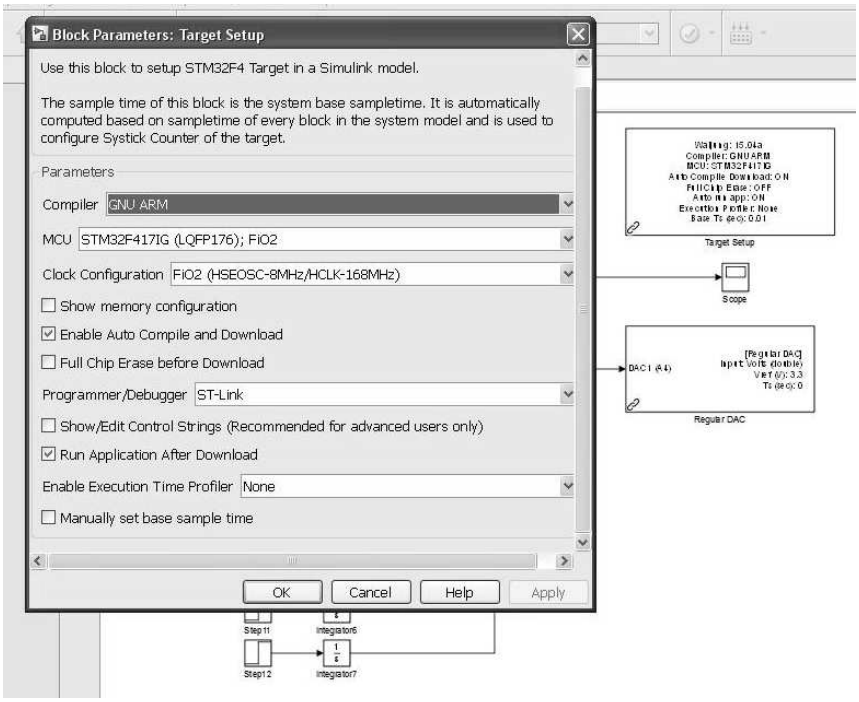


Рисунок 7.6 – Налаштування блоку Target Setup у моделі Matlab Simulink

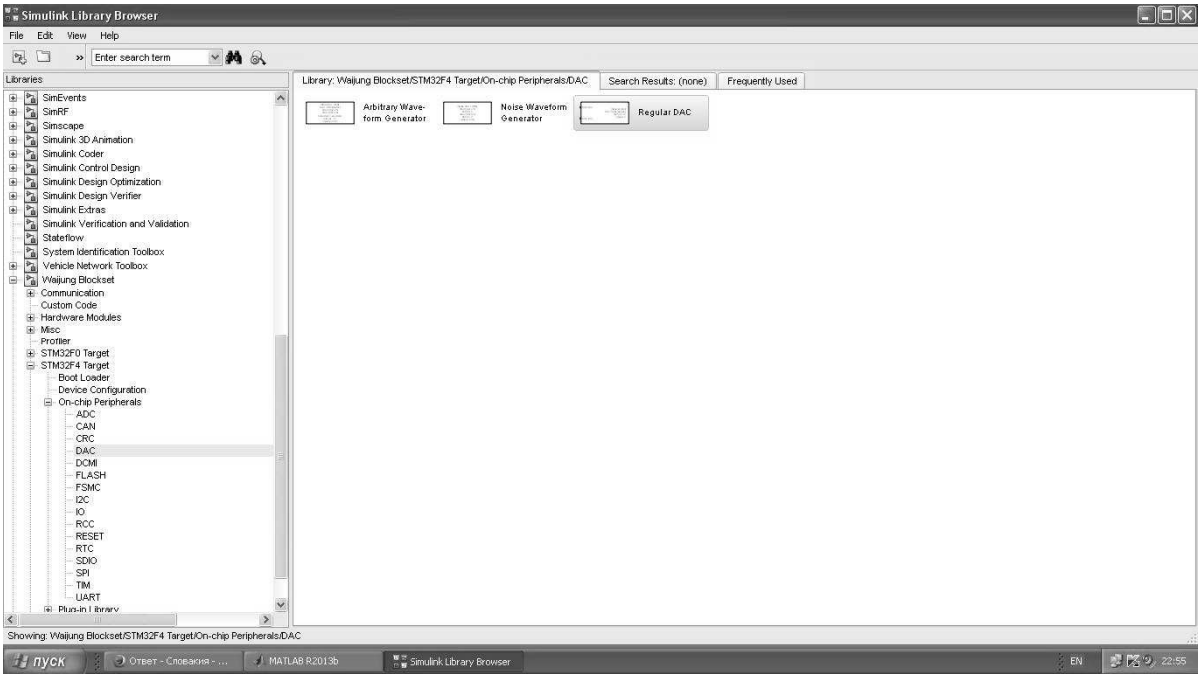


Рисунок 7.7 – Розташування блоку Regular DAC у бібліотеці Simulink Library Browser

Налаштування блоку цифро-аналогового перетворювача наведено на рис. 7.8.

Результати моделювання роботи цифро-аналогового перетворювача в Matlab Simulink і осциллограмма вихідного сигналу цифро-аналогового перетворювача на виході порту A PA4 мікроконтролера STM32F4Discovery наведено на рис. 7.9 (а, б), відповідно.

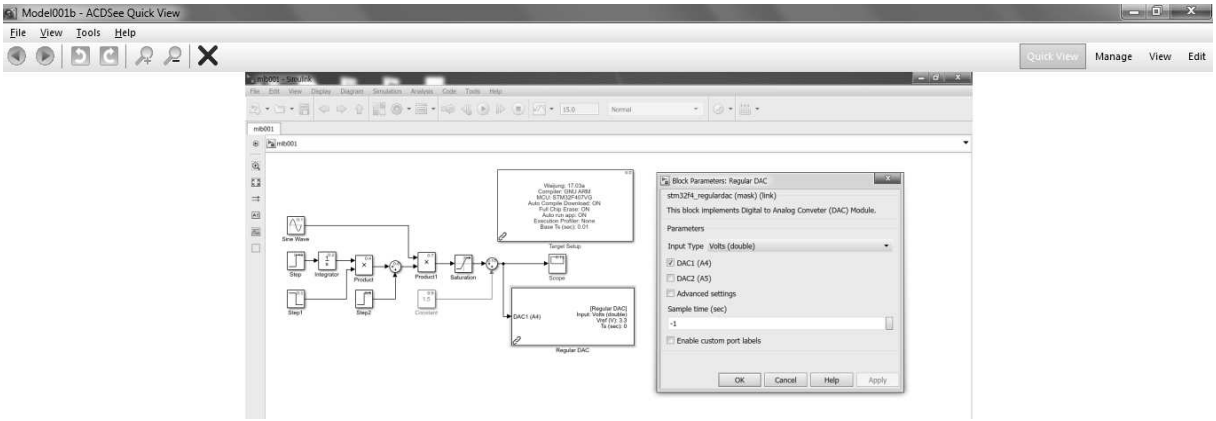
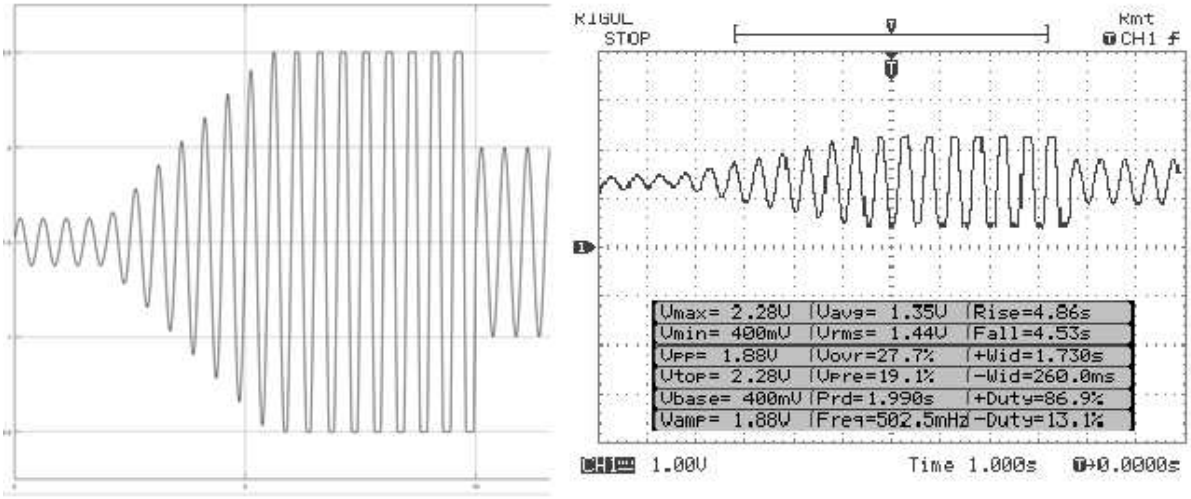


Рисунок 7.8 – Налаштування блоку Regular DAC у моделі Matlab Simulink



а б

Рисунок 7.9 – Графік сформованого сигналу на виході цифро-аналогового перетворювача в Matlab Simulink (а), осцилограма сформованого сигналу на виході PA4 мікроконтролера STM32F4Discovery (б)

Таким чином, використовуючи Matlab Simulink для програмування мікроконтролерів, можна розробляти програми будь-якої складності без знання мови програмування С. Створена модель перетвориться системою Matlab Simulink на С-код у вигляді окремих модулів. Також створюється бінарний файл прошивки з розширенням BIN. Цей файл можна завантажити в мікроконтролер за допомогою спеціалізованої утиліти для прошивання, наприклад ST-Link.

Однак для оптимізації скомпільованого коду, повного використання можливостей мікроконтролера STM32F4Discovery знання мови програмування С не тільки бажано, але й необхідно.

РОЗДІЛ 8
ІНТЕРФЕЙС SPI У STM32

SPI (англ. Serial Peripheral Interface, SPI bus – послідовний периферійний інтерфейс, шина SPI) – послідовний синхронний стандарт передавання даних у режимі повного дуплексу, призначений для забезпечення простого й недорогого поєднання мікроконтролерів і периферії. SPI також іноді називають чьотирьохпровідним (англ. four-wire) інтерфейсом.

Як і будь-який інший інтерфейс, SPI використовується для передавання даних від одного пристрою до іншого. Пристрої на шині SPI не рівноправні. Як правило, є один головний пристрій (Master) і безліч під порядкованих пристроїв (Slave). Зазвичай у ролі майстра виступає мікроконтролер, а підпорядкованими пристроями є різна периферія: термодатчики, акселерометри, годинники реального часу тощо. Майстер називається так, тому що без його відома жоден підпорядкований пристрій не робитиме жодного обміну даними. Сама шина SPI фізично являє собою 4 проводи [22]:

- MOSI – з цього проводу дані йдуть від Master- до Slave-пристрою;
- MISO – з цього проводу дані йдуть від Slave- до Master-пристрою;
- SCK – через цей провід Master передає тактовий сигнал до Slave-пристроїв;
- CS (Chip select або SS – Slave select) – через цей провід майстер дає зрозуміти підпорядкованому пристрою, що зараз він пересилає дані саме йому.

З опису всіх чотирьох ліній можна зробити висновки:

SPI – це послідовний інтерфейс: біти даних передаються один за іншим;

SPI – це синхронний інтерфейс: передавання даних (у будь-яку сторону) відбувається тільки в той час, коли майстер генерує імпульси синхронізації й передає їх через провід SCK інших пристроїв на шині.

До однієї шини може підключатися кілька пристроїв, кількість яких теоретично не обмежена.

SPI надає можливість обмінюватися даними в повнодуплексному режимі. Поки майстер генерує тактові імпульси, він може посилати дані на підпорядкований пристрій і одночасно приймати їх від нього ж.

На рисунку 8.1 розглянуто підключення пристроїв до шини; стрілки показують, який пристрій і куди передає сигнал.

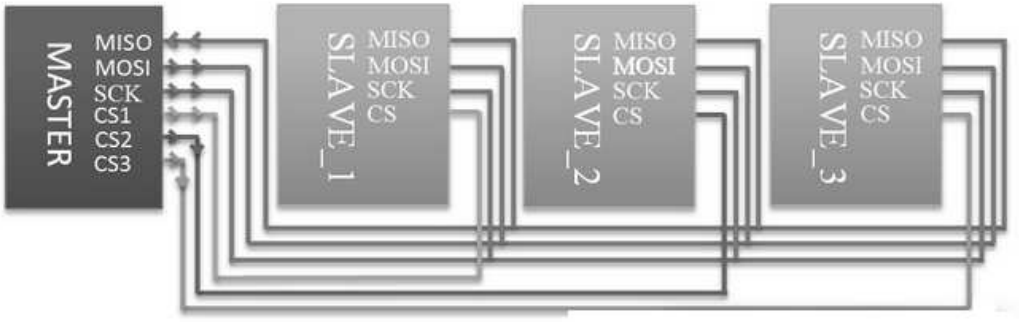


Рисунок 8.1 – Структурна схема передавання даних

З рис. 8.1 видно, що всі лінії інтерфейсу, крім CS, просто об'єднуються між собою. Для кожного підпорядкованого пристрою майстер має окремий вихід CS. Для того щоб обмінятися даними з іншим підпорядкованим пристроєм, майстер встановить на виході CS2 низький логічний рівень, а на двох інших (CS1 і CS3) – високий.

Таким чином, Slave_1 і Slave_3 не будуть задіяні й тим самим не створять перешкод спілкуванню майстра і Slave_2.

Активний стан контакту CS – це логічний нуль. У такої схеми є один недолік: на 10 підпорядкованих пристроїв майстер повинен мати 10 окремих виводів для підключення до CS. Існує інший варіант підключення, який називається daisy-chain. При такому підключенні всі пристрої з'єднуються в ланцюжок і мають один загальний CS. Детально цей спосіб підключення розглядатися не буде з огляду на те, що використовується він досить рідко. Як уже згадувалося вище, до однієї шини можуть бути підключені різні Slave-пристрої, деякі з них досить швидкі й можуть обмінюватися даними з майстром на великій швидкості, а деякі, навпаки, дуже повільні. Це означає, що майстер не повинен надто швидко генерувати тактові імпульси, в іншому випадку повільні slave-пристрої його не сприймуть через спотворення даних.

Синхронізація в SPI

Частота проходження бітових інтервалів у лініях передавання даних визначається синхросигналом SCK, який генерує провідний пристрій, ведені пристрої використовують синхросигнал для визначення моментів зміни бітів на лінії даних, при цьому ведені пристрої ніяк не можуть впливати на частоту проходження бітових інтервалів. Як у провідному пристрої, так і в підпорядкованому є лічильник імпульсів синхронізації (бітів). Лічильник у підпорядкованому пристрої дозволяє останньому визначити момент закінчення передавання пакета. Лічильник скидається при вимкненні підсистеми SPI. Така можливість завжди є в провідному пристрої. У веденому пристрої лічильник зазвичай скидається деактивацією інтерфейсного сигналу SS.

Оскільки дії провідного й веденого пристроїв тактуються одним і тим самим сигналом, то до стабільності цього сигналу не ставляться ніякі вимоги, за винятком обмеження тривалості напівперіодів, яка визначається максимальною робочою частотою більш повільного пристрою. Це дозволяє використовувати SPI в системах з низькою стабільною тактовою частотою, а також полегшує програмну емуляцію провідного пристрою.

РОЗДІЛ 9
ОСОБЛИВОСТІ ПРОГРАМУВАННЯ АЦП І ЦАП МІКРОКОНТРОЛЕРА STM32F4

Для коректної роботи вбудованого аналого-цифрового (АЦП) і цифро-аналогового (ЦАП) перетворювачів, вбудованих у мікроконтролер STM32F4Discovery, потрібно його правильно програмно налаштувати. У цьому розділі наведено особливості налаштування АЦП і ЦАП, вбудованих у мікроконтролер. Докладно описано функції зі стандартної бібліотеки STM32F4Discovery. Неправильне налаштування параметрів АЦП або ЦАП може бути причиною появи некоректних даних вимірювань, або повної непрацездатності ЦАП або АЦП. Тому інформація, що міститься в цьому розділі, є актуальною при вивченні програмування мікроконтролерів STM32F4Discovery.

Блок-схема алгоритму налаштування наявного в мікроконтролері АЦП і його порту наведено на рис. 9.1.

Рисунок 9.1 – Блок-схема алгоритму налаштування АЦП

Для того щоб налаштувати вбудований аналого-цифровий перетворювач, необхідно налаштувати порт (у цьому прикладі використовується PA1) так, щоб він працював як аналоговий вхід АЦП. Під час налаштування виходу порту PA1 на вхід необхідно програмно вибрати режим аналогового входу. На рис. 9.2 наведено програмний код процедури налаштування й ініціалізації входу-виходу порту для роботи в режимі аналогового входу АЦП ADC1.

```
void adc_pin_init() {

      GPIO_InitTypeDef gpio;

      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
      GPIO_StructInit(&gpio);
      gpio.GPIO_Mode = GPIO_Mode_AN;
      gpio.GPIO_Pin = GPIO_Pin_1;
      GPIO_Init(GPIOA, &gpio);
      }
```

Рисунок 9.2 – Налаштування режиму роботи виводу порту PA1

У цій процедурі спочатку відбувається увімкнення порту за допомогою команди RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE). Далі вибирається режим аналогового входу (gpio.GPIO_Mode = GPIO_Mode_AN), після чого реалізується застосування структури gpio для виконання налаштувань порту.

Після виконання процедури налаштування порту виконується налаштування режиму АЦП ADC1 для того, щоб він міг здійснювати вимірювання й конвертування сигналу в цифрову форму.

На рис. 9.3 наведено процедуру ініціалізації ADC1, яка здійснює його налаштування.

```
// Настраиваем АЦП

      void adc_init() {

            ADC_InitTypeDef ADC_InitStructure;
            ADC_CommonInitTypeDef adc_init;
            RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
            ADC_DeInit();
            ADC_StructInit(&ADC_InitStructure);
            adc_init.ADC_Mode = ADC_Mode_Independent;
            adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
            ADC_InitStructure.ADC_ScanConvMode = DISABLE;
            ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
            ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConvEdge_None;
            ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
            ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
            ADC_CommonInit(&adc_init);
            ADC_Init(ADC1, &ADC_InitStructure);
            ADC_Cmd(ADC1, ENABLE);
            }
```

Рисунок 9.3 – Процедура ініціалізації АЦП

Спочатку на ADC1 подаються тактові імпульси від шини APB2 (команда RCC_APB2PeriphClockCmd (RCC_APB2Periph_ADC1, ENABLE)). Тобто АЦП підключений до джерела живлення (за умовчанням з метою зменшення енергоспоживання всі периферійні пристрої мікроконтролера STM32F4Discovery вимкнені). Команда ADC_DeInit () ініціює скидання всіх налаштовань. Це виконується для перестраховування при ініціалізації. Далі вибирається незалежний режим роботи ADC1 (ADC_Mode_Independent). Тобто робота ADC1 не залежить від інших АЦП мікроконтролера.

За допомогою команд ADC_InitStructure.ADC_ScanConvMode = DISABLE і ADC_InitStructure.ADC_ContinuousConvMode = DISABLE вибирається режим роботи нетривалого й одиночного перетворення ADC1. Команда ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None дозволяє вибрати джерело запуску перетворень регулярної групи каналів АЦП (у цьому випадку не використовується). Ним може бути будь-яке переривання або зовнішня подія. ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right вирівнює дані по правому краю. ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b вибирає розділення АЦП 12 біт.

Для того щоб АЦП ADC1 міг здійснювати будь-які вимірювання й перетворення, необхідно його увімкнути (ADC_Cmd (ADC1, ENABLE)). Інакше він просто не буде функціонувати.

Існують інжектовані й регулярні канали АЦП. Результат перетворень АЦП з регулярних каналів зберігається в одному регістрі.

Інжектовані канали зберігають результати перетворень в окремі регістри. В цьому випадку використовуються регулярні канали.

Для перетворення аналогового сигналу на вході АЦП розроблена спеціальна функція для зчитування й конвертування аналогового сигналу в цифрову форму (рис. 9.4).

```
// Функция чтения данных из АЦП

    u16 readADC1(u8 channel) {
          ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_3Cycles);
          ADC_SoftwareStartConv(ADC1);
          while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
          return ADC_GetConversionValue(ADC1);
          }
```

Рисунок 9.4 – Функція зчитування й конвертування аналогового сигналу в цифрову форму

Команда ADC_RegularChannelConfig (ADC1, channel, 1, ADC_SampleTime_3Cycles) виконує конфігурування регулярних каналів АЦП. При конфігуруванні ADC1 вибирається його канал АЦП, який визначається змінною channel і налаштовується час вибирання. Від параметра ADC_SampleTime залежить точність виконуваних перетворень.

Функція ADC_SoftwareStartConv (ADC1) дозволяє перетворення для ADC1. У циклі while (ADC_GetFlagStatus (ADC1, ADC_FLAG_EOC) == RESET) очікується закінчення запущеного процесу конвертування величини аналогового сигналу в цифрову форму.

Рядок return ADC_GetConversionValue (ADC1) повертає поточне значення величини оцифрованого сигналу на вході.

Функція readADC1(), яка викликається з основної програми, здійснює процес перетворення аналогового сигналу на вході АЦП у цифрову форму.

Мікроконтролер STM32F407VG включає в себе два вбудовані ЦАП. Розрядність ЦАП складає 12 біт. ЦАП (DAC) використовує в якості виходів порт А: PA4 – DAC1, PA5 – DAC2.

Фрагмент програмного коду на мові С для формування вихідного сигналу пилкоподібної напруги на виході ЦАП наведено на рис. 9.5.

```
// амплитудные значения ЦАП для построения пилообразного напряжения
const uint16_t signal[40]= {
        0,  100,  200,  300,  400,  500,  600, 700,
        800, 900, 1000, 1100, 1200, 1300, 1400, 1500,
        1600, 1700, 1800, 1900, 2100, 2200, 2300, 2400,
        2500,  2600,  2700,  2800,  2900,  3000,  3100,  3200,
        3300,  3400,  3500,  3600,  3700,  3800,  3900,  4000,
         };

unsigned char i=0;

// Обработчик прерывания от таймера 6
void TIM6_DAC_IRQHandler(void) {

  TIM_ClearITPendingBit(TIM6, TIM_IT_Update); // Сбрасываем флаг
обработки прерывания от таймера

  DAC_SetChannel2Data(DAC_Align_12b_R,signal[i++]); // Записываем в
ЦАП очередной элемент массива и равняем по правому краю
  if (i==40) i=0;
}
```

Рисунок 9.5 – Формування пилкоподібної напруги на виході ЦАП за допомогою програми на мові С

Таким чином, при виникненні переривання по переповненнютаймера TIM6 у регістр ЦАП завантажується наступне по порядку значення з масиву для формування вихідного сигналу.

Фрагмент програмного коду налаштування таймера TIM6 і цифро-аналогового перетворювача DAC (другий канал) наведено на рис. 9.6.

```
// ---------------------------- ЦАП ----------------------------------------//

      // Тактирование ЦАП
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

      // Тактирование таймера TIM6
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,ENABLE);

      // Настройка таймера TIM6
      TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;
      TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
      TIM_TimeBaseStructure.TIM_Prescaler     = 8400-1;
      TIM_TimeBaseStructure.TIM_Period        = 5000-1; // частота перебора
значений массива
      TIM_TimeBaseStructure.TIM_CounterMode   = TIM_CounterMode_Up;
      TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);
      TIM_Cmd(TIM6, ENABLE);

      TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE); // включаем прерывания по
переполнению
      NVIC_EnableIRQ(TIM6_DAC_IRQn);             //Разрешение TIM6_DAC_IRQn
прерывания

      // Включаем DAC2
      DAC_Cmd(DAC_Channel_2, ENABLE);

//-------------------------------------------------------------------------//
```

Рисунок 9.6 – Налаштування TIM6 і DAC

Таким чином, для активізації роботи таймера TIM6 і DAC подаються тактові імпульси, як і для будь-якого-іншого периферійного пристрою мікроконтролера STM32F4Discovery:

RCC_APB1PeriphClockCmd (RCC_APB1Periph_DAC, ENABLE);
RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM6, ENABLE);

Далі налаштовуються переддільник таймера і його період (час, через який буде виникати переривання). Для цього використовуються такі рядки програмного коду:

TIM_TimeBaseStructure.TIM_Prescaler = 8400-1;
TIM_TimeBaseStructure.TIM_Period = 5000-1;
Вибирається режим рахування таймера (рахування вгору):
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
Дозволяється переривання по переповненню таймера TIM6:
TIM_ITConfig (TIM6, TIM_IT_Update, ENABLE);
NVIC_EnableIRQ (TIM6_DAC_IRQn);
Для функціонування DAC його необхідно увімкнути:
DAC_Cmd (DAC_Channel_2, ENABLE);

РОЗДІЛ 10
ПРИКЛАД РОЗРОБЛЕННЯ РЯДКОВОГО ПРОТОКОЛУ ПЕРЕДАВАННЯ ДАНИХ МІЖ ПЕРСОНАЛЬНИМ КОМП'ЮТЕРОМ І МІКРОКОНТРОЛЕРОМ

Сучасні засоби автоматизації в більшості випадків використовують програмовані елементи різної функціональної насиченості й складності. Як правило, невеликі системи керування будуються навколо єдиного керівного ядра, яким є мікроконтролер або мікропроцесор. У разі масштабних рішень розробники стоять перед вибором або будувати всю систему керування на базі одного високопродуктивного ядра, яке має всю необхідну периферію й великий простір введення-виведення, або ж реалізувати розподілену систему. Важливим елементом у побудові розподілених систем є організація середовища передавання даних, а також принципів доступу до нього, що приводить до пошуку існуючих мережевих рішень або розроблення власних протоколів обміну даними. Серед існуючих протоколів найбільш широко відомі CAN, ProfiBus, FieldBus, що працюють на фізичному рівні відповідно до стандарту RS-485 [23]. Їх відмінними рисами є висока надійність, насичена функціональність, можливість виявлення й коригування помилок. Недоліком є використання як фізичного середовища мідної крученої пари, яка схильна до паразитних наведень з боку зовнішніх електромагнітних полів.

Послідовний канал, порівняно з контролером, повільний. На швидкості 9600 бод передавання одного символу займає більше мілісекунди. Тому, коли контролер щільно завантажений обчисленнями й не повинен їх зупиняти на час обміну по UART, потрібно використовувати переривання по завершенню приймання й передавання символу. Можна виділити місце в пам'яті для формування посилки на передавання й збереження прийнятої посилки (буфер посилки), а також покажчики на позицію поточного символу. Переривання по завершенню приймання або передавання символу викликають відповідні підпрограми, які передають або зберігають черговий символ із зсувом покажчика й перевіркою ознаки кінця повідомлення, після чого повертають керування основній програмі до наступного переривання. По завершенню надсилання й прийняття всієї посилки або формується користувальницький прапор, що відпрацьовується в основному циклі програми, або відразу викликається підпрограма оброблення повідомлення.

Огляд існуючих протоколів передавання даних

У загальному випадку передавання по послідовному каналу складається з керівних байтів (синхронізації передавання, адреси відправника й одержувача, контрольної суми тощо) і власне байтів даних.

Основне завдання в організації протоколу – забезпечити розрізнення керівних байтів і байтів даних на мікроконтролері. Наприклад, ведений пристрій, отримуючи по лінії потік байтів, має визначати, де «початок» посилки й де «кінець» і якому пристрою адресоване передавання даних.

Протокол MODBUS працює за принципом «запит – відповідь» між пристроями. Цей протокол дуже популярний і широко використовується в промисловості. Однак він має ряд надлишкових можливостей, які дуже рідко застосовуються на практиці. Для його застосування потрібно опанувати складну систему команд протоколу й писати власний драйвер для його роботи [24]. Тому найбільший інтерес для роботи з мікроконтролером STM32F4Discovery можуть становити протоколи на основі ASCII-коду.

Керівні символи й дані передаються у вигляді звичайних ASCII-символів. Посилка може виглядати так:

у HEX-вигляді: 3Ah 31h 32h 52h 53h 34h 38h 35h 0Dh;

в ASCII-вигляді: ":" "1" "2" "R" "S" "4" "8" "5" / ПС /.

На початку керівний символ початку посилки ":", наступні дві цифри – адреса одержувача (1, 2), потім – символи даних (RS485) і в кінці – керівний символ кінця посилки переведення рядка – / ПС / (0Dh). Усі пристрої на лінії, прийнявши символ ":", починають записувати в пам'ять посилку до символу кінця рядка – 0Dh. Потім порівнюють адресу з посилки зі своєю адресою. Пристрій з цією адресою обробляє дані посилки, решта – ігнорують посилку. Дані можуть містити будь-які символи, крім керівних (":", 0Dh).

Перевага таких протоколів полягає в зручності налаштовування системи та простоті синхронізації посилок. Можна через перетворювач RS485-RS232 підключити лінію до COM-порту комп'ютера та в будь-якій термінальній програмі побачити всю інформацію. що проходить, у зручному вигляді (у вигляді рядків) [25].

Недоліком є відносно великий розмір посилки при передаванні великої кількості двійкової інформації, адже на передавання кожного байта потрібно два ASCII-символи (7Fh – "7", "F") [26; 27].

Далі йтиметься про власне рядковий протокол передавання даних (ASCII-протокол). У якості мікропроцесорного пристрою використовується 32-розрядний мікроконтролер STM32F4 фірми STMicroelectronics. Для з'єднання ноутбука або комп'ютера, який не має стандартного послідовного порту COM, можна використовувати недорогий перетворювач-конвертор USB-UART (універсальний асинхронний приймач). Цей перетворювач при підключенні до USB-порту комп'ютера та встановлення спеціальних драйверів емулює послідовний порт. Таким чином, операційна система буде «бачити» цей перетворювач як звичайний COM-порт.

Структура пропонованого рядкового протоколу передавання даних

Протокол буде використовуватися для передавання 11 параметрів, для керування підпорядкованим пристроєм (мікроконтролером). Для того щоб розрізнити керівні байти й байти даних, буде використаний розділь-ник у вигляді ASCII-символу. Оскільки при прийманні й передаванні на лінії можуть виникнути проблеми з цілісністю даних, до протоколу включено перевірку контрольної суми, яка обчислюється на мікроконтролері й

на стороні головного пристрою (ПК, ноутбук). Для відсилання й приймання даних розроблено спеціалізований додаток на мові програмування C# у відкритому середовищі розроблення SharpDevelop. Також до протоколу включено байт номера пристрою, якому адресується посилка.

Для зв'язку головного й підпорядкованого пристрою розроблено схему підключення (рис. 10.1).

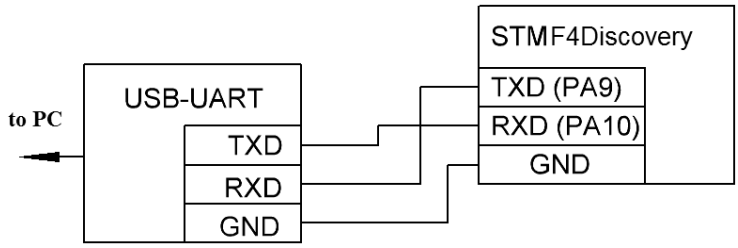


Рисунок 10.1 – Схема підключення обладнання

Структура посилок микроконтролеру (рис. 10.2) і назад на ПК має вигляд, наведений на рис. 10.3.

#	№Device	#	#	Param1	#	#	…	#	#	Param11	#	#	CRC	#

Рисунок 10.2 – Структура запиту на мікроконтролер

– роздільник керівних байтів і байтів з даними;

№Device – керівний байт, містить номер пристрою, якому адресується посилка;

Param1...Param11 – передані 11 параметрів (дані);

CRC – байти контрольної суми для перевірки цілісності даних.

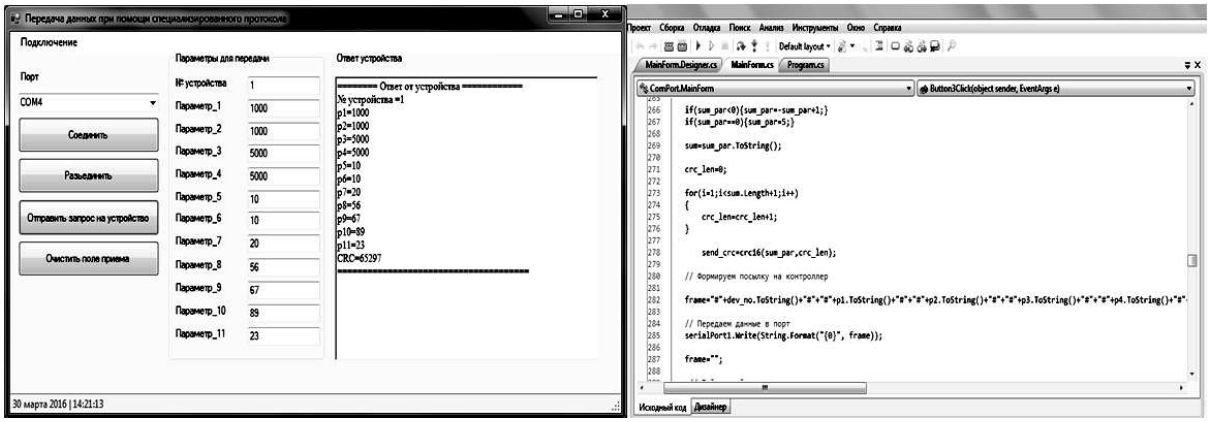
#	№Device	#	Param1	#	…	#	Param11	#	CRC	#

Рисунок 10.3 – Структура відповіді мікроконтролера

Розроблений протокол може бути реалізований із використанням будь-якої мови програмування. У якості демонстрації роботи протоколу було розроблено спеціалізований додаток для передавання параметрів мікроконтролеру й отримання відповіді від нього. Відповіддю є значення параметрів, які були передані на мікроконтролер (рис. 10.4).

Реалізація протоколу на мікроконтролері STM32F4Discovery і фрагменти програмного коду на мові програмування C наведено на рис. 10.5.

Для з'єднання ноутбука або комп'ютера, який не має стандартного послідовного порту COM, можна використовувати недорогий перетворювач-конвертор USB-UART (універсальний асинхронний приймач рис. 10.6). Цей перетворювач при підключенні до USB-порту комп'ютера й установлення спеціальних драйверів емулює послідовний порт.



а *б*

а – інтерфейс розробленого додатка; б – вікно програмного коду
Рисунок 10.4 – Реалізація протоколу на стороні ПК (C # SharpDevelop)

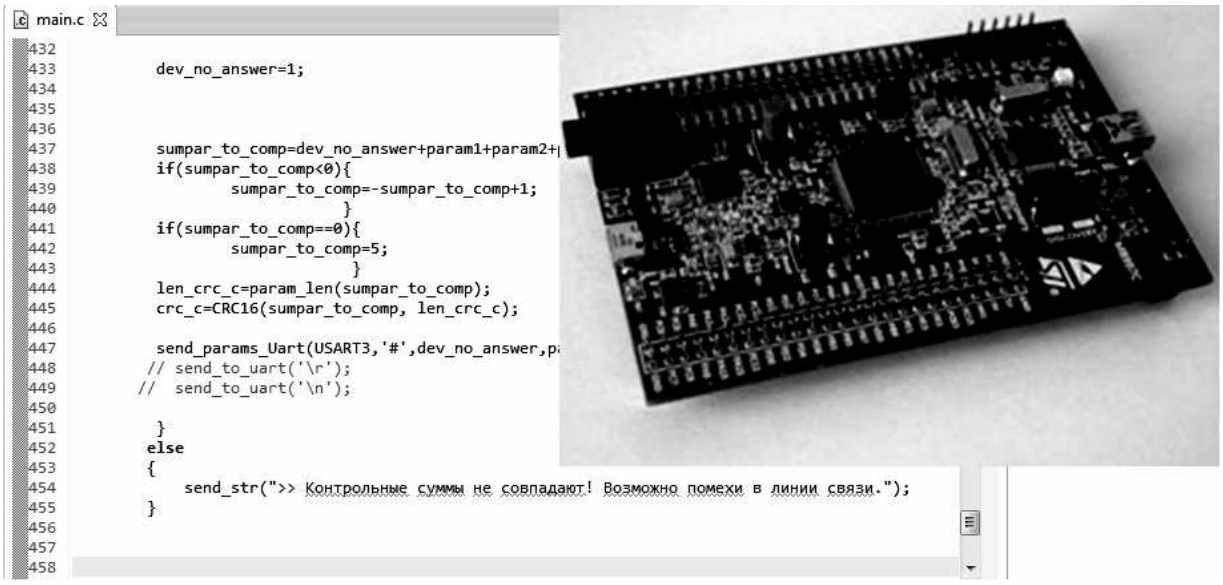


Рисунок 10.5 – Реалізація протоколу на боці мікроконтролера STM32F4 і його зовнішній вигляд

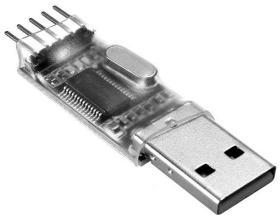


Рисунок 10.6 – Зовнішній вигляд USB-UART-перетворювача

Цей протокол був також реалізований на боці персонального комп'ютера на мові Object Pascal для перевірки адекватності його роботи (рис. 10.7, 10.8).

Таким чином, був розроблений власний простий протокол передавання даних між головним (ПК) і підпорядкованим (мікроконтролер) пристроями. Цей протокол дозволяє перевірити коректність одержуваних даних за допомогою розрахунку контрольної суми на боці мікроконтролера і комп'ютера. Цей протокол простий, надійний у використанні й реалізації.

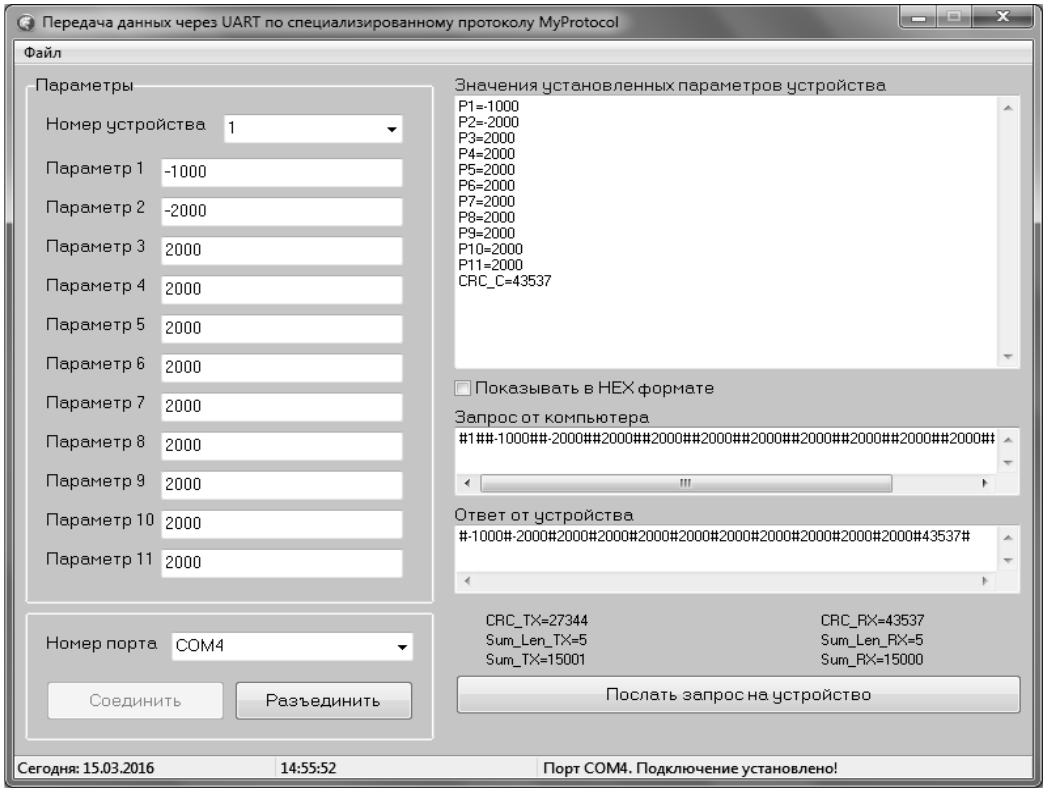


Рисунок 10.7 – Демонстрація роботи розробленого протоколу (інтерфейсу)

```
//  Процедура парсирования полученных данных

procedure input_data_parse(var p:string; var protocol:string; var position:integer);
var i:integer;
begin
  for i:=position to length(protocol) do
  begin
  if (protocol[position]='#') then
   begin
   if (protocol[i]='#') and (i<>position) then
   begin
    position:=i;
    break;
   end;
     if(i>position) then
      p:=p+protocol[i];
   end;
  end;
end;

//  Функция расчета контрольной суммы

function crc16(data:integer;len:integer):integer;
var crc:integer; index:integer;
begin
        crc:=$FFFF;


        while (len<>0) do
```

Рисунок 10.8 – Реалізація протоколу на боці ПК (фрагмент програмного коду Pascal)

РОЗДІЛ 11
ОСОБЛИВОСТІ ФОРМУВАННЯ СКЛАДНИХ СИГНАЛІВ ЗА ДОПОМОГОЮ ШІМ

Шляхом зміни коефіцієнта заповнення регістра CCRx таймера мікроконтролера STM32F4Discovery, який служить для генерування ШІМ-сигналу при незмінному періоді таймера можна отримати модульований сигнал абсолютно будь-якої форми. Буде розглянуто формування сигналу синусоїдальної форми частоти 50 Гц. Воно здійснюється окремими напівперіодами, тобто один напівперіод виводиться на один канал, інший напівперіод – на інший канал ШІМ-таймера. Формування відбувається по перериванню при переповненні таймера. У регістр «захоплення – порівняння» CCRx завантажуються значення масиву синусоїди по перериванню. Масив синусоїдального сигналу складається з 40 точок на напівперіод. Значення цього масиву обчислені заздалегідь. Формування ШІМ-синусоїди здійснюється за допомогою таймера TIM1. Розрахунок періоду таймера відбувається так: оскільки частота синусоїди 50 Гц, то, відповідно, період 0,02 с. Оскільки сигнал формується за напівперіодами, то значення напівперіоду складає 0,01 с. Напівперіод містить 40 точок для формування синусоїди, тому значення напівперіоду ділиться на 40. Значення періоду таймера – 0,00025 с. Далі потрібно налаштувати період таймера TIM1. Максимальне значення CCR для формування синусоїди – 250. Шляхом завантаження в CCRx значення масиву синусоїди по перериванню від таймера TIM1 по переповненню (при досягненні значення періоду таймера ARR) виходить модульований сигнал синусоїди на двох каналах. На одному каналі – позитивна напівхвиля, на іншому – негативна (канали TIM1 CH3 і CH4, відповідно, виводи порту A PA8, PA10). Оброблювач переривання буде викликатися через кожні 0,00025 секунди.

Результати формування синусоїдального сигналу за допомогою ШІМ (ШІМ-сигнал і усереднені значення [11] напруги сигналу) при різних коефіцієнтах заповнення наведено на рис. 11.1 (а, б) у вигляді графіка залежності вихідної напруги від часу [28].

Фрагмент оброблювача переривань, де власне й відбувається занесення значень з масиву синусоїди в регістри захоплення-порівняння CCR1 і CCR3, і, відповідно, формування синусоїдального сигналу за допомогою зміни скважності імпульсів при кожному виклику переривання (рис. 11.2) [29].

Масиви даних для формування синусоїдального ШІМ-сигналу на виходах мікроконтролера наведено на рис. 11.3.

Загальний алгоритм здійснення налаштувань портів введення-виведення, таймерів для формування вихідного ШІМ-сигналу наведено на рис. 11.4.

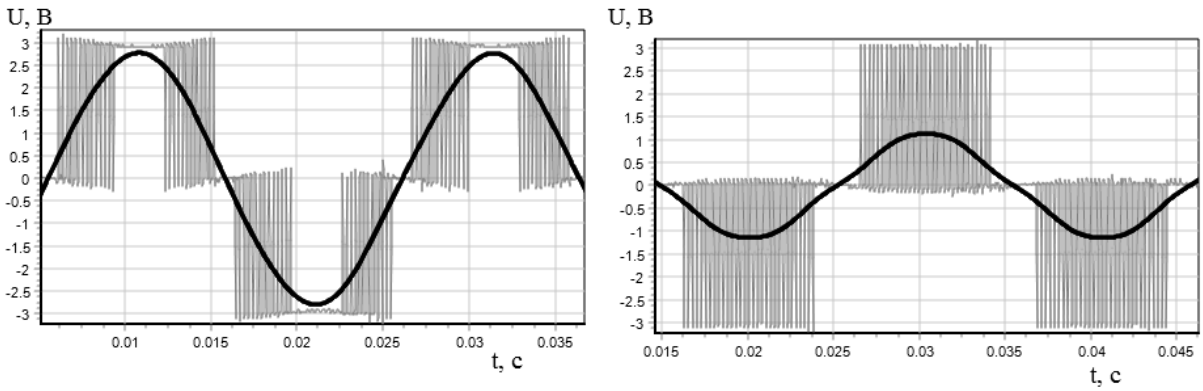


Рисунок 11.1 – Залежності напруги вихідного сигналу ШІМ і його усередненого значення від часу при різних коефіцієнтах заповнення (а, б)

```
// Обработчик прерываниия по переполнению таймера TIM1
void TIM1_UP_TIM10_IRQHandler(void) {
if (TIM_GetITStatus(TIM1, TIM_IT_Update) != RESET) {
TIM_ClearITPendingBit(TIM1, TIM_IT_Update);


TIM1->CCR1 = k*sinarr[i];
TIM1->CCR3 = k*sinarrinv[i];


i++;

if(i==82) {i=0;}
}
}
```

Рисунок 11.2 – Оброблювач переривання по переповнення таймера TIM1

```
36 const uint32_t sinarr[82] = {
37
38         0,20,39,58,77,96,113,131,147,162,177,190,202,213,223,231,238,243,247,249,250,249,247,
39         243,238,231,223,213,202,190,177,162,147,131,113,96,77,58,39,20,0,0,0,0,0,0,0,0,0,0,0,
40         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
41 };
42
43 const uint32_t sinarrinv[82] = {
44
45         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,
46         0,20,39,58,77,96,113,131,147,162,177,190,202,213,223,231,238,243,247,249,250,249,
47         247,243,238,231,223,213,202,190,177,162,147,131,113,96,77,58,39,20,0
48
49 };
```

Рисунок 11.3 – Масив даних для формування ШІМ-синусоїди

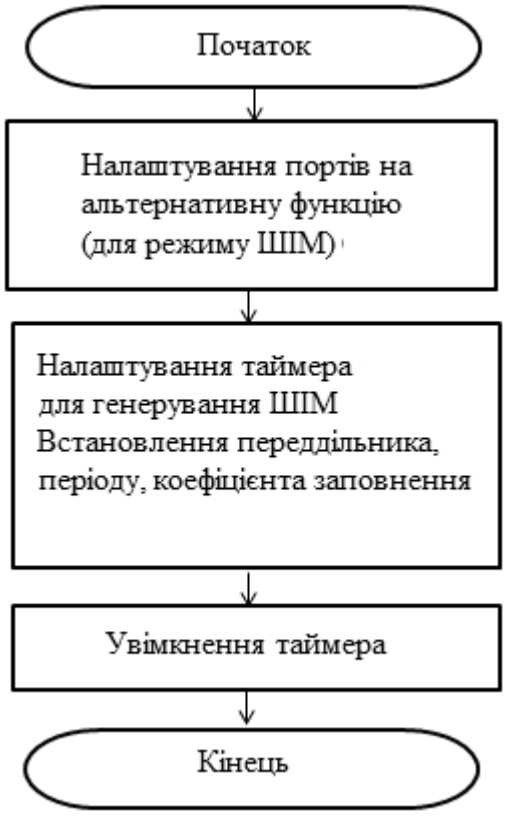


Рисунок 11.4 – Алгоритм налаштування периферійних пристроїв для формування ШІМ (портів введення-виведення, таймерів)

Для живлення установок електроприводу змінного струму й у системах електроживлення різних споживачів широко використовують інвертори напруги, коректори коефіцієнта потужності й активні подовжувачі. Для забезпечення високої якості електричної енергії на вході й (або) виході таких перетворювачів застосовують різні види широтно-імпульсного модулювання (ШІМ).

Широтно-імпульсне модулювання (pulse-width modulation) – імпульсний сигнал постійної частоти й змінної скважности, тобто відношення періоду проходження імпульсу до його тривалості.

За допомогою задавання скважності (тривалості імпульсів) можна змінювати середню напругу на виході ШІМ.

З появою швидкодійної потужної мікропроцесорної техніки принцип ШІМ активно застосовується в сучасних частотних перетворювачах для отримання трифазної синусоїдальної напруги змінних частоти й амплітуди, для керування швидкістю двигуна постійного струму, у різних інверторах, де потрібно отримати з випрямленого сигналу змінний бажаних частоти, амплітуди й форми.

Використання ШІМ дозволяє змінювати амплітуду випрямленої напруги за допомогою регулювання шпаруватості імпульсів.

Як ключі для здійснення перемикань випрямленої напруги використовуються біполярні, польові, швидкодійні IGBT-транзистори. Вони дозволяють здійснювати перемикання з високою частотою. Таким чином, за допо-

могою перемикань за певним законом можна з вихідного сигналу випрямленої напруги отримати вихідний сигнал потрібних форми й частоти [30].

За допомогою ШІМ можна формувати модульований сигнал практично будь-якої форми з використанням сучасних мікроконтролерів.

Існує ШІМ аналогова й цифрова. Аналогова використовується в аналогових електронних системах і вважається класичною. Однак на сьогодні широкого поширення набула цифрова ШІМ, тому що вона активно використовується мікроконтролерами для формування сигналів різних форми й частоти.

ШІМ підрозділяється на однополярне й двополярне. У однополярного ШІМ немає негативного сигналу. Однополярне джерело живлення має тільки позитивний потенціал (+ V) і сигнал землі (GND).

Більшість сучасних мікроконтролерів мають однополярне живлення й видають на виходах однополярні сигнали.

ВИСНОВКИ

У виконаній роботі розглянуто далеко не всі можливості мікроконтролера STM32F4. Практичне розроблення отримали лише ті напрямки, які в першу чергу були цікаві авторам для вирішення конкретних завдань. Не розглянутими залишилися можливості використання вбудованого акселерометра, відтворення медіафайлів, які містить оцінна плата STM32F4Discovery і які за потреби буде розглянуто в наступних роботах.

Автори монографії висловлюють надію, що подана робота може дійсно бути гарною базою при знайомстві й самостійному вивченні програмованих мікроконтролерів STM32 бакалаврами, магістрами спеціальностей КІТ, АВП, ЕСА, а також для аспірантів, інших розробників, проектувальників і наукових працівників, які розробляють нові машини й технології. Інформацію про всі виявлені зауваження, доповнення та помічені недоліки просимо надсилати на адресу babashandrey@gmail.com.

ПЕРЕЛІК ПОСИЛАНЬ

1. *Квашнін В. О.* Методологія програмування мікроконтролерів Stm32F4Discovery і практичного їх застосування для вирішення наукових та інженерних задач / В. О. Квашнін, А. В. Бабаш, В. В. Квашнін // Сучасна освіта – доступність, якість, визнання : збірник наукових. – Краматорськ : ДДМА, 2016. – 209 с.

2. RISC [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/RISC.

3. Оценочная плата STM32F4 Discovery с STM32F407 [Электронный ресурс]. – URL: http://robotosha.ru/stm32/stm32f407-discovery-board.html.

4. Язык программирования Си [Электронный ресурс]. – URL: http://lib.ru/MAN/DEMOS210/c.txt.

5. *Керниган Б.* Язык программирования Си : пер. с англ. / Керниган Б., Ритчи Д. – 3-е изд., испр. – СПб : Невский Диалект, 2001. – 352 с.

6. STM32 → Порты GPIO [Электронный ресурс]. – URL: http://catethysis.ru/stm32-gpio/.

7. Порты микроконтроллера [Электронный ресурс]. – URL: http://easystm32.ru/for-beginners/11-mcu-ports.

8. *Бугаев В. И.* Лабораторный практикум для изучения микро-контроллеров архитектуры ARM Cortex-M4 на базе оценочного модуля STM32F4Discovery / В. И. Бугаев, М. П. Мусиенко, Я. М. Крайных. – Москва : Николаев : МФТИ ; ЧГУ, 2013. – 71 с.

9. DM00031020 Reference manual [Электронный ресурс]. – URL: http://www.st.com/st-web ui static/active en/resource/technical/document/ reference_manual / DM00031020.pdf.

10. DM00039084 User manual [Электронный ресурс]. – URL:http://www.st.com/st-web.

11. Stm32. 4. последовательный порт (uart) [Электронный ресурс]. – URL: http://alex-exe.ru/radio/stm32/stm32-uart-spl/.

12. UART в STM32. Часть 1 [Электронный ресурс]. – URL: http://easystm32.ru/interfaces/15-uart-in-stm32-part-1.

13. UART в STM32. Часть 2 [Электронный ресурс]. – URL: http://easystm32.ru/interfaces/16-uart-in-stm32-part-2.

14. *Осипов Н. А.* Разработка приложений на C# / Н. А. Осипов. – СПб : НИУ ИТМО, 2012. –118 с.

15. *Квашнин В. О.* Использование встроенных цифро-аналогового и аналого-цифрового преобразователей микроконтроллера Stm32f4Discovery / В. О. Квашнин, А. В. Бабаш // Научный вестник ДГМА. – 2016. – № 1 (19е). – С. 47–58.

16. STM32: генератор синусоидального сигнала на TIM и DMA [Электронный ресурс]. – URL: http://catethysis.ru/stm32-tim-dma-pwm-sin.

17. Обитель RC-инженера [Электронный ресурс]. – URL: http://vg.ucoz.ru/publ/programmirovanie_mikrokontrollerov_stm32/stm32f4_cap_dac_po_vzroslomu/9-1-0-28.

18. STM32. General-Purpose Timers. Режим захвата [Электронный ресурс]. – URL: http://www.galaktika.zp.ua/news/407-stm32-general-purpose-timers-r%D0%B5zi%D0%BC-z%D0%B0hv%D0%B0t%D0%B0.html

19 Stm32 захват значения таймера [Электронный ресурс]. – URL: radiokot.ru/forum/viewtopic.php?p=1864755.

20. Matlab Simulink application to program microcontrollers STM32F4Discovery // Vladyslav Kvashnin, Babash Andrey, Valeriy Kvashnin, Yuliya Lazutkina,, Galina Klimenko // Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering Nitra. – 2016. – P. 134–140.

21. *Квашнин В. В.* Программирование микроконтроллеров STM32f4Discovery в Matlab Simulink / В. В. Квашнин, А. В. Бабаш, Г. П. Клименко // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : тези доповідей VIII Міжнародної науково-практичної конференції (21–23 вересня 2016 р., м. Запоріжжя). – Запоріжжя : ЗНТУ, 2016. – С. 287–289.

22. Интерфейс SPI в STM32. Часть 1 [Электронный ресурс]. – URL: http://easystm32.ru/interfaces/43-spi-interface-part-1.

23. *Babash A.* Proprietary data transfer protocol between personal computers and microcontrollers STM32F4DISCOVERY development method / Andrey Babash, Valeriy Kvashnin, Alexander Tarasov // Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering Nitra. – 2016. – P. 30–35.

24. *Бабаш А. В.* Способы защиты передаваемых данных с использованием специализированного протокола / А. В. Бабаш, В. О. Квашнин, А. Ф. Тарасов // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : тези доповідей VIII Міжнародної науково-практичної конференції (21–23 вересня 2016 р., м. Запоріжжя). – Запоріжжя : ЗНТУ, 2016. – С. 229–231.

25. *Матвеев Д. А.* Разработка протокола микроконтроллеры сетей в оптическом кольце через UART [Электронный ресурс] / Матвеев Д. А. – URL: http://fetmag.mrsu.ru/2010-2/pdf/NetworkProtocol.pdf.

26. Modbus Application Protocol Specification [Электронный ресурс]. – URL: www.Modbus-IDA.org.

27. Протокол передачи RS-485 [Электронный ресурс]. – URL: http://estohard.narod.ru/Protocols/rs485/rs485_1.htm.

28. *Квашнин В. О.* Методика определения динамических скоростной и токовой характеристик асинхронного электропривода / В. О. Квашнин, А. В. Бабаш // Электротехнические и компьютерные системы. – 2015. – № 19 (95). – С. 28–32.

29. *Kvashnin V.* Generating PWM by using microcontroller Stm32F4disovery / V. Kvashnin, A. Babash // Electrotechnic and computer systems. – 2016. – № 22 (98). – P. 277–283. – URL: ui/static/active/en/ resource/technical/document/user_manual/DM00039084.pdf.

30. *Чаплыгин Е. Е.* Инверторы напряжения и их спектральные модели : учебное пособие / Е. Е. Чаплыгин. – М. : МЭИ, 2003. – С. 30.

Наукове видання

КВАШНІН Валерій Олегович,

БАБАШ Андрій Владиславович,

КВАШНІН Владислав Валерійович

ПРОГРАМУВАННЯ ТА ЗАСТОСУВАННЯ МІКРОКОНТРОЛЕРІВ STM32F4DISCOVERY

Монографія

Редагування, комп’ютерне верстання О. М. Болкова

83/2017. Формат 60 × 84/16. Ум. друк. арк. 8,37.
Обл.-вид. арк. 7,56. Тираж 300 пр. Зам. № 1566.

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Академічна, 72.
Свідоцтво суб’єкта видавничої справи
ДК №1633 від 24.12.2003

Виготівник
ЦТРІ «Друкарський дім»
84306, м. Краматорськ, вул. Олекси Тихого, 1-б,
тел.: (06264) 6-73-34, (066) 076-76-21.
Свідоцтво суб’єкта видавничої справи
ДК № 5071 від 23.03.2016