МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
МІКРОПРОЦЕСОРНІ СИСТЕМИ

Навчальний посібник

Рекомендувала Науково-методична рада
Національного університету “Львівська політехніка”

Львів
Видавництво Львівської політехніки
2021

УДК 004
П 88

Рецензенти:

Рак Т. Є., доктор технічних наук, доцент, професор кафедри інформаційних технологій, проректор "ІТ СТЕП Університет";

Кремінь В. Т., кандидат технічних наук, доцент, Senior Principal Engineer, фірма "Infineon Technologies";

Парамуд Я. С., кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка"
як навчальний посібник для студентів галузі знань 12 "Інформаційні технології"
спеціальності 123 "Комп'ютерна інженерія"
(протокол № 59 від 20 жовтня 2021 р.)

Пуйда В. Я.

П 88 Лабораторний практикум з дисципліни "Мікропроцесорні системи": навч. посібник / В. Я. Пуйда. – Львів: Видавництво Львівської політехніки, 2021. – 218 с. – Режим доступу: http://eom.lp.edu.ua/textbooks/np-mps.pdf вільний. – Заголовок з екрана.

ISBN 978-966-941-690-2

Практикум містить комплекс з шести методичних вказівок до виконання лабораторних робіт з дисципліни "Мікропроцесорні системи".

УДК 004

ISBN 978-966-941-690-2

ВСТУП

Мікропроцесори – елементна база більшості сучасних комп'ютерних систем, які ще називають мікропроцесорними системами (МПС). Мікропроцесорні компоненти – це сучасні мікроелектронні засоби для обробки інформації, представленої в дискретній формі, по заданому алгоритму, реалізованому у вигляді програми. Базовим елементом мікропроцесорної техніки є мікропроцесор (МП). Мікропроцесори появилися завдяки розвитку мікроелектронної промисловості. Архітектурно перші мікропроцесори повторювали архітектуру центрального процесора універсальної електронної обчислювальної машини і реалізувалися на одному чи декількох мікроелектронних компонентах. Завдяки дешевизні, низькій ціні, можливості забезпечення масового випуску мікропроцесорних компонентів, використання методів обробки дискретної інформації отримало широке розповсюдження не тільки в наукових застосуваннях та в промисловості, а і в самих широких сферах людської діяльності включаючи побут та сферу розваг. Сьогодні важко знайти область де не застосовується мікропроцесорна техніка.

Мікропроцесори використовуються в універсальних персональних комп'ютерах, в спеціалізованих комп'ютерних системах для наукових досліджень, на виробництві, в побуті та інших сферах, наприклад, в системах керування технологічними процесами, інформаційно – вимірювальних системах, технічних системах спеціального призначення (військова техніка, авіація, бортові комп'ютери тощо), системах зв'язку, побутовій техніці.

В 1970 році було випущено перший серійний мікропроцесор для військових цілей F14CADC (почато освоєння і розсекречено в 1968 році). Intel 4004 було випущено 15 листопада 1971 року – це перший масовий мікропроцесор, який був розроблений і виготовлений корпорацією Intel. Головними розробниками процесора були Федеріко Фаґґін і Тед Гофф. Архітектурно це був центральний процесор на одному кристалі з варіантом Гарвардської архітектури, яка використовувала розділені пам'ять програм та даних розміщених на окремих зовнішніх кристалах. Об'єднання вузлів центрального процесора та зовнішньої пам'яті здійснено одною мультиплексованою 4-бітовою шиною через необхідність мінімізації числа виводів на корпусі мікросхеми. Операційний пристрій та пристрій керування забезпечують реалізацію 46 машинних команд. Набір регістрів містить 16 регістрів по 4 біти кожен. Максимальна тактова частота – 740 кГц. Після успішного використання першого мікропроцесора фірма Intel почала розробку та випуск нових складніших та потужніших мікропроцесорів. Власне завдяки мікропроцесорам фірми Intel появилися перші персональні комп'ютери.

Успіхи у використанні перших мікропроцесорів спонукали ряд фірм почати розробку власних мікропроцесорів або випускати клони мікропроцесорів фірми Intel. В Україні на ВО "Кристал", а пізніше і на інших підприємствах України також освоїли клони деяких мікропроцесорів фірми Intel – I8080A(КР580ВМ80А), I8086 (КР1810ВМ86), I8088(КР1810ВМ88).

Для студентів спеціальності 123 "Комп'ютерна інженерія" має велике значення формування знань та навиків в області мікропроцесорної техніки, отримання систематизованих знань про напрямки використання мікропроцесорної техніки, розвиток мікропроцесорної техніки, освоєння сучасних мікропроцесорних компонентів та методів і засобів проєктування МПС, оволодіння методикою проектування мікропроцесорних систем різного функціонального призначення.

Лабораторний практикум з дисципліни “Мікропроцесорні системи” допоможе здобувачеві освіти отримати знання про теоретичні основи використання мікропроцесорної техніки в обчислювальних та керуючих пристроях, функціонально-орієнтованих МПС, освоїти архітектуру та технічні і конструктивні характеристики сучасних типових мікропроцесорних компонентів, навчитися вибирати мікропроцесорні компоненти для забезпечення заданих технічних характеристик при проєктуванні функціонально-орієнтованої МПС, отримати навики по вибору та практичному оволодінню навиками використання програмних середовищ розроблення програмного забезпечення для функціонально-орієнтованих МПС, навчитися вибирати та оволодіти навиками використання типових відлагоджувальних модулів та спеціальних апаратних засобів в процесі розроблення та налагодження функціонально-орієнтованих МПС.

Перелік робіт лабораторного практикуму

№ з/п	Назви та зміст лабораторних робіт
1	**“Дослідження інтегрованого на кристалі мікроконтролера аналого-цифрового перетворювача ”**. Основні режими роботи аналого-цифрового перетворювача (АЦП). Реалізація основних етапів відлагодження драйвера АЦП.
2	**“Дослідження вузла виведення графічної інформації в МПС".** Графічні індикатори. Формування сигналів керування. Реалізація основних етапів відлагодження драйвера виводу графічної інформації в МПС.
3.	**“Дослідження вузла введення відеоінформації в МПС”.** Аналогові та цифрові відеокамери. Основні режими функціонування. Формування сигналів керування. Реалізація основних етапів відлагодження драйвера введення відео- інформації в МПС.
4	**“Дослідження вузла введення в МПС інформації з цифрового сенсора”.** Цифрові сенсори температури, тиску, загазованості. Реалізація основних етапів відлагодження драйвера введення в МПС інформації з цифрового сенсора.
5	**“Дослідження режимів керування компонентами МПС з допомогою паралельного та послідовного інтерфейсів”.** Ознайомитися з організацією паралельних портів, інтерфейсом I2C, організацією інтерфесу RS-485, Flash FM24CL16, керуванням світлодіодами та реле. Реалізація основних етапів відлагодження драйвера керування компонентами МПС.
6	**“Дослідження цифрового синтезатора частоти в МПС”.** Основні режими роботи інтегрованого на кристалі мікроконтролера цифрового синтезатора частоти. Реалізація основних етапів відлагодження драйвера цифрового синтезатора частоти.

Лабораторна робота № 1

ДОСЛІДЖЕННЯ ІНТЕГРОВАНОГО НА КРИСТАЛІ МІКРОКОНТРОЛЕРА АНАЛОГО-ЦИФРОВОГО ПЕРЕТВОРЮВАЧА

МЕТА РОБОТИ: дослідити режими функціонування інтегрованого на кристалі мікроконтролера аналого-цифрового перетворювача (АЦП) з використанням Visual Studio 2019 Community; ознайомитися з підлюченням до мікропроцесорної системи (МПС) та функціонуванням символьного LCD індикатора та інтерфейсу UART.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з ядром, структурою, вузлами АЦП, паралельних портів та UART мікроконтролера **STM32F407VG** (*STM32F407xx.pdf*).
2. Ознайомитися з лабораторним стендом на основі ***на відлагоджувального модуля*** **STM32F4Discovery** (*STM32F4DISCOVERY User manual.pdf*)**.**
3. Перевірити свою теоретичну підготовку по контрольних питаннях для самоперевірки; за необхідності скористатися методичними матеріалами та літературними джерелами.
4. Запустити **Visual Studio 2019 Community**.
5. Ознайомитись з елементами екрану відлагоджувача, використанням команд головного меню та підменю, режимами відображення завантаженої програми, операціями модифікації вмісту комірок пам'яті та регістрів.
6. Ознайомитися з прикладом програми.
7. Виконати основні етапи процесу налагодження програми:
   - створити новий проєкт або завантажити тестовий проєкт;
   - налагодити програму у різних режимах відлагоджувача;
   - запустити програму **ModbusFS 2.2.**
8. Виконати індивідуальне завдання визначене керівником лабораторної роботи.
9. Захистити лабораторну роботу.

ЗАВДАННЯ

1. Створити проєкт в **Visual Studio 2019 Community**.
2. В покроковому режимі продемонструвати роботу драйвера АЦП.
3. В покроковому режимі продемонструвати роботу драйвера символьного LCD.
4. В покроковому режимі продемонструвати роботу драйвера UART.
5. Зняти характеристику АЦП, використовуючи COM-порт та програму **ModbusFS 2.2** (термінал COM-порту).

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Внутрішня структура мікроконтролера STM32F407VG.
2. Програмна модель ядра Cortex-F4.
3. Програмна модель мікроконтролера STM32F407VG.

4. АЦП мікроконтролера STM32F407VG.
5. Паралельні порти мікроконтролера STM32F407VG.
6. Порт UART мікроконтролера STM32F407VG.
7. Структура відлагоджувального модуля STM32F4Discovery.
8. Основні опції меню відлагоджувача.

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

1. Запустити **Visual Studio 2019 Community**.
2. Ознайомитися за допомогою інтерактивної довідки, яка входить у склад програмного пакету, з можливостями середовища та його функціями, з елементами графічного інтерфейсу та з режимами відлагодження.
3. Виконати основні етапи процесу налагодження програми в системі mVision:

- **створити** новий проєкт **Embedded** або **завантажити** тестовий; порядок створення нового проєкту див. скріншоти нижче;
- налагодити програму у різних режимах.

4. Запустити програму **ModbusFS2.2** (термінал COM-порту).

ОСНОВНІ ЕТАПИ створення проєкту Embedded та налагодження програми у Visual Studio 2019 Community

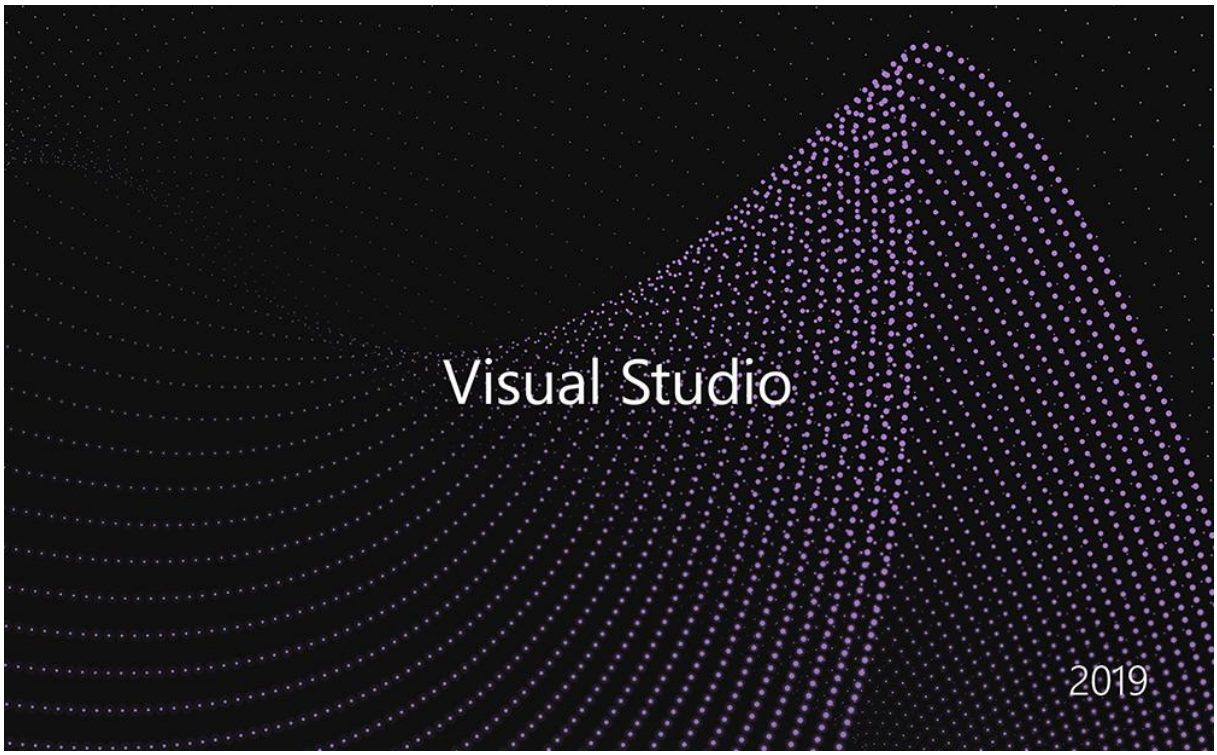


*Рис. 1.1. Стартове вікно **Visual Studio 2019 Community***

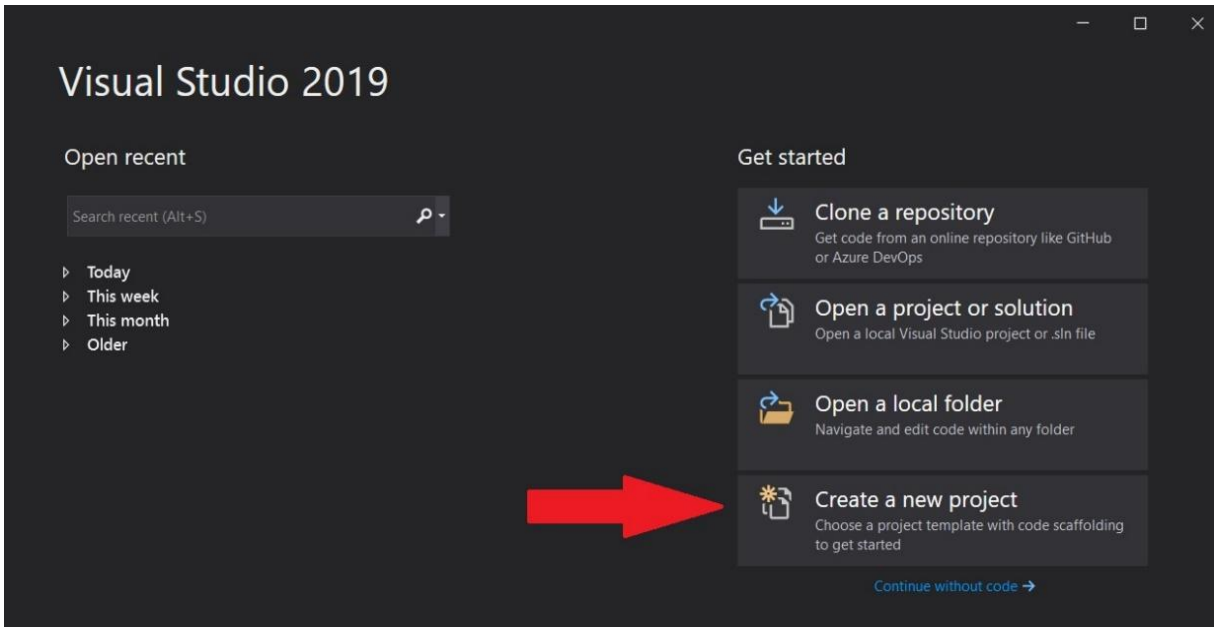


Рис. 1.2. Створення нового проєкту

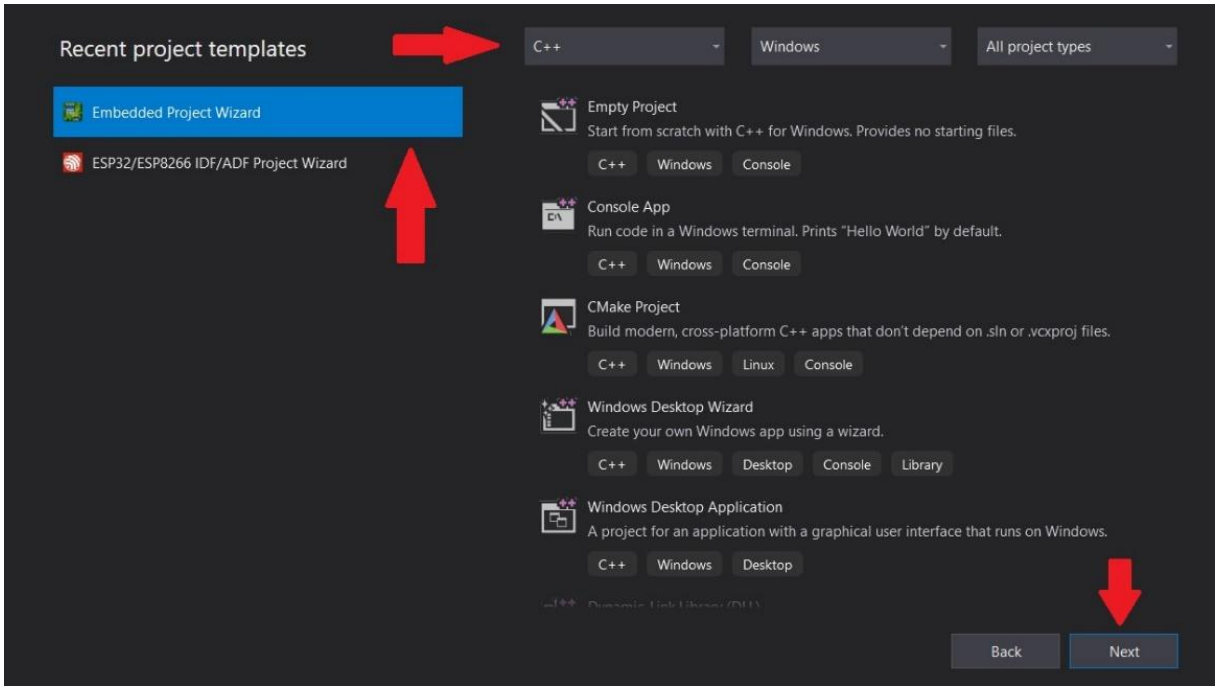


*Рис. 1.3. Вибір типу проєкту **Embedded** та мови програмування*

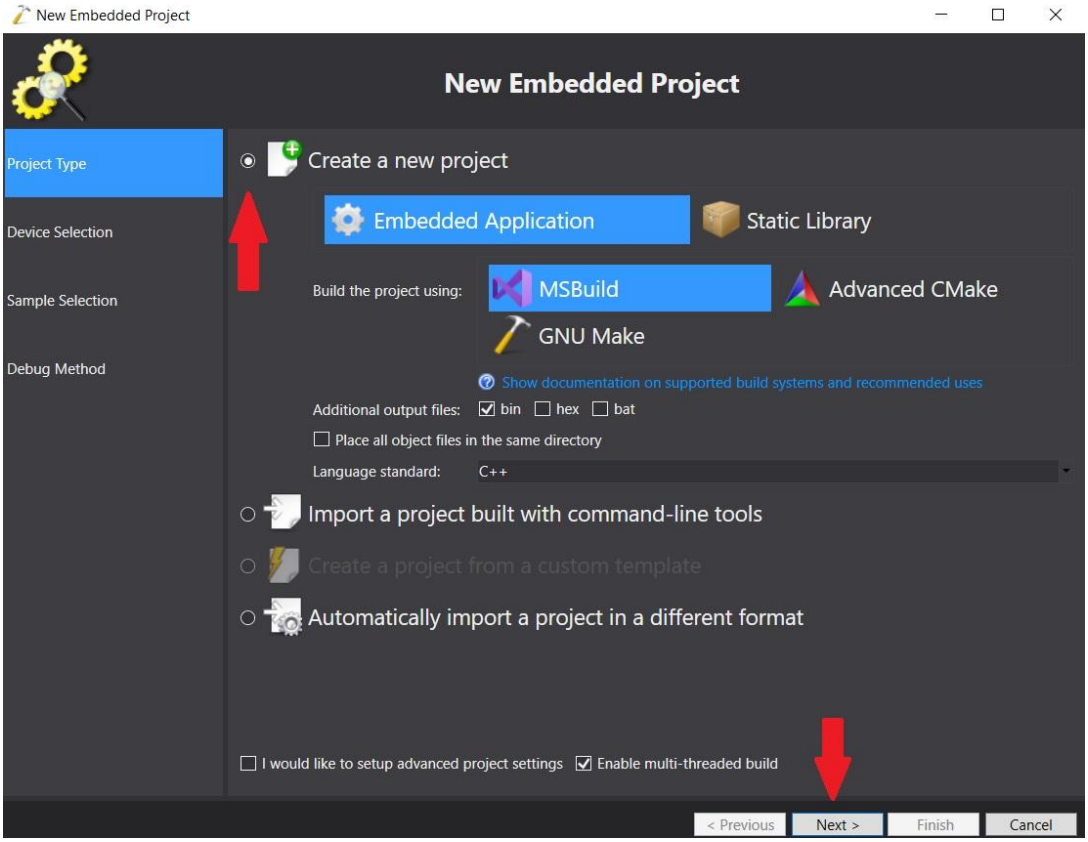


Рис. 1.4. Вибір розміщення та назви проєкту

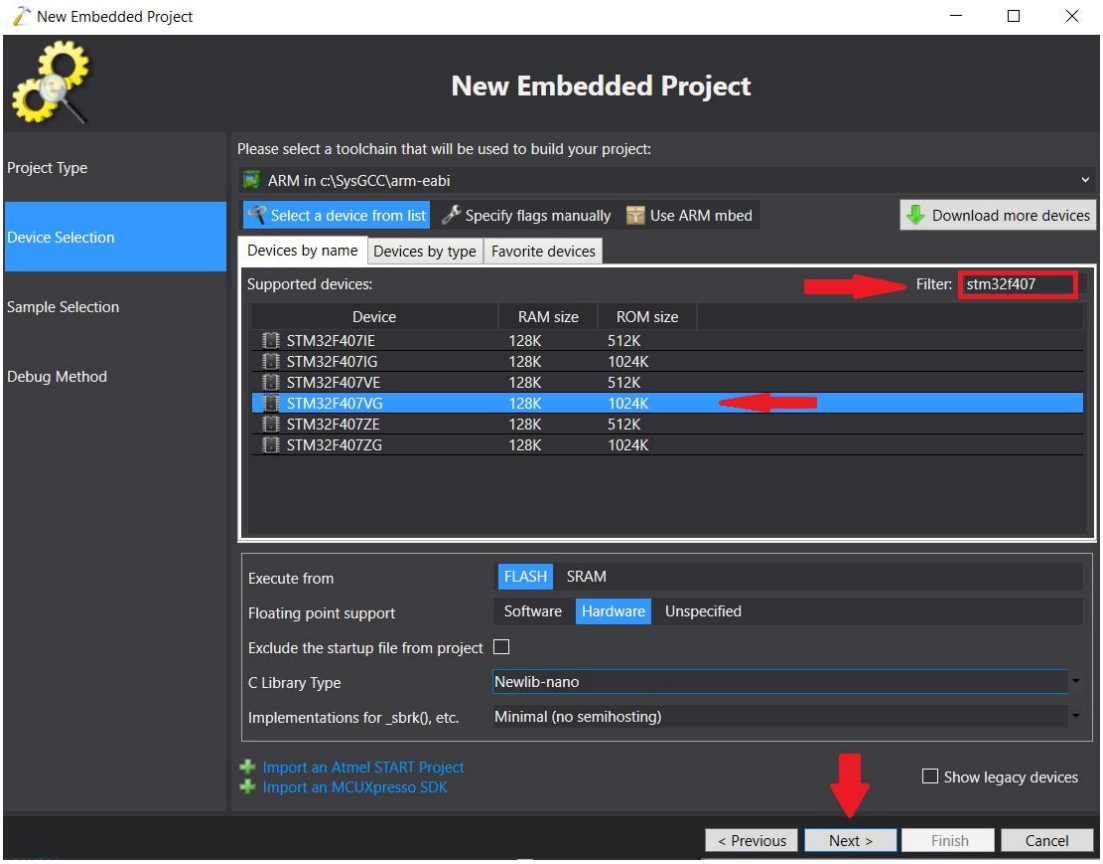


Рис. 1.5. Вибір типу мікроконтролера (STM32F407VG)

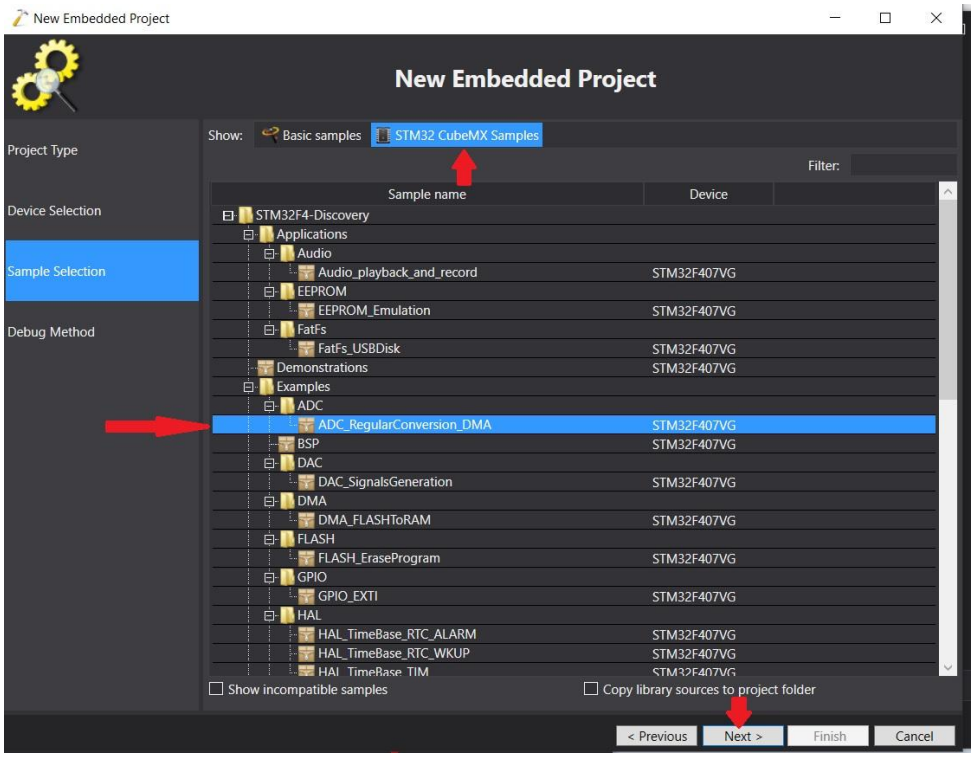


Рис. 1.6. Вибір проєкту з переліку прикладів

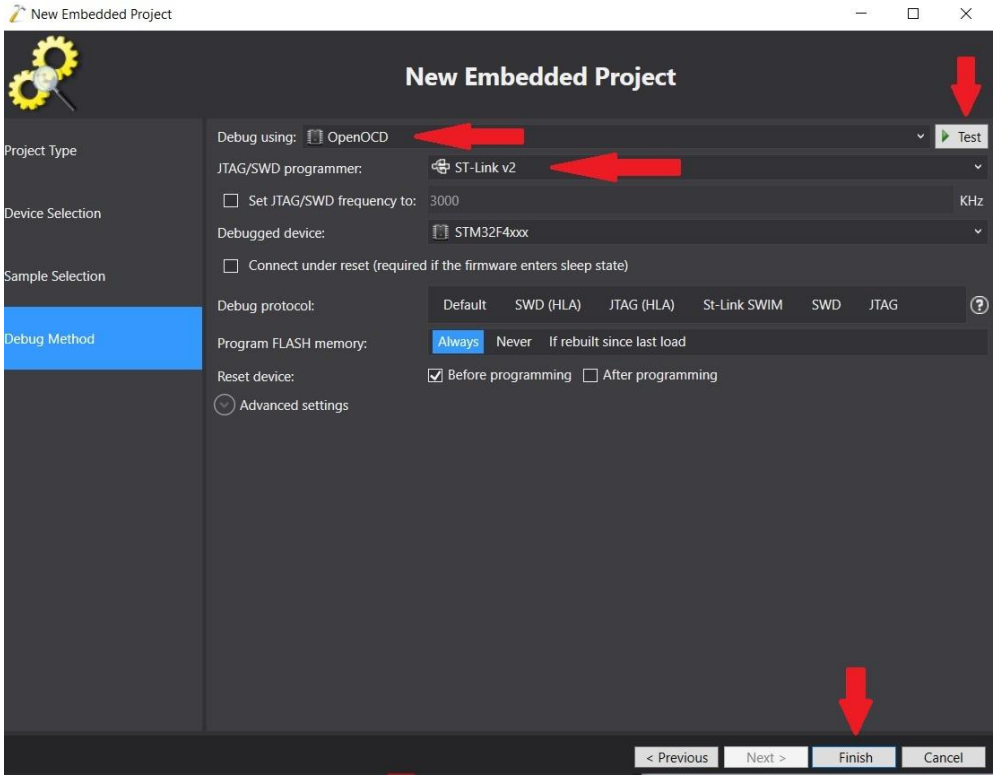


Рис. 1.7. Вибір режимів апаратного відлагоджувача

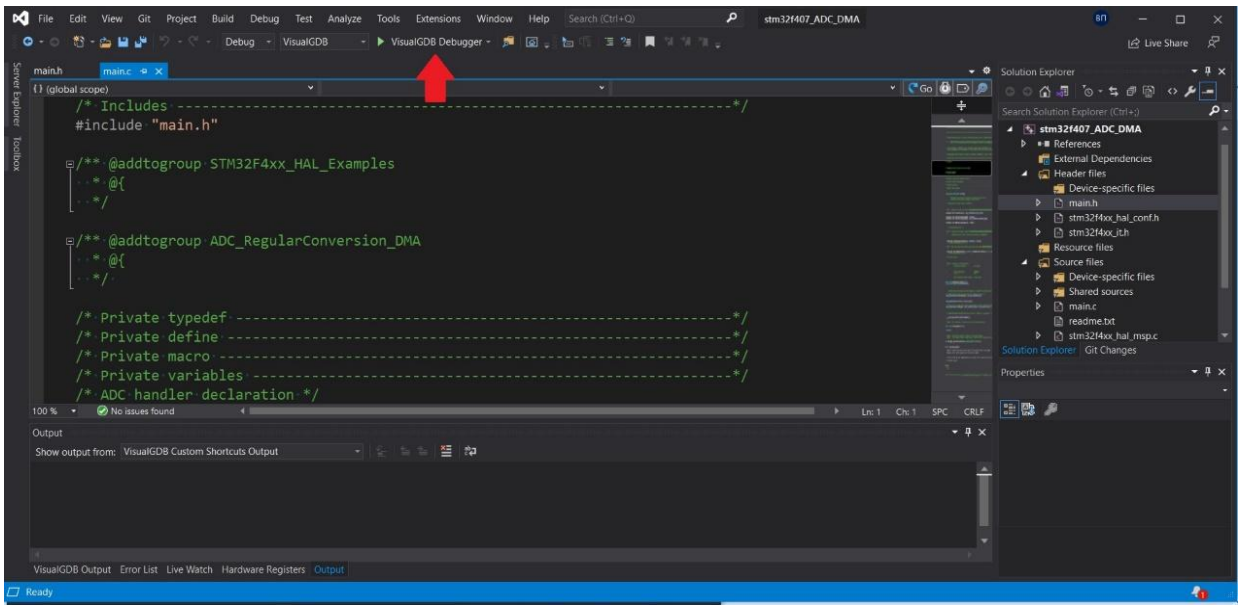


Рис. 1.8а. Запуск режиму компілювання та відлагодження програми, встановлення точки переривання

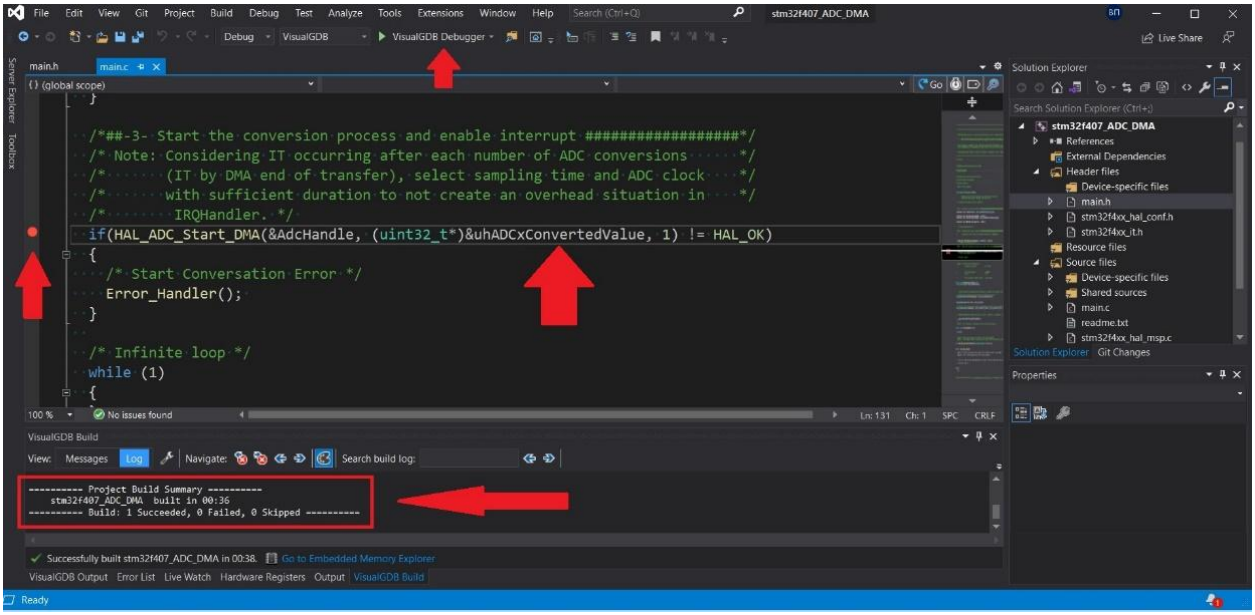


Рис. 1.8б. Запуск режиму компілювання та відлагодження програми, встановлення точки переривання

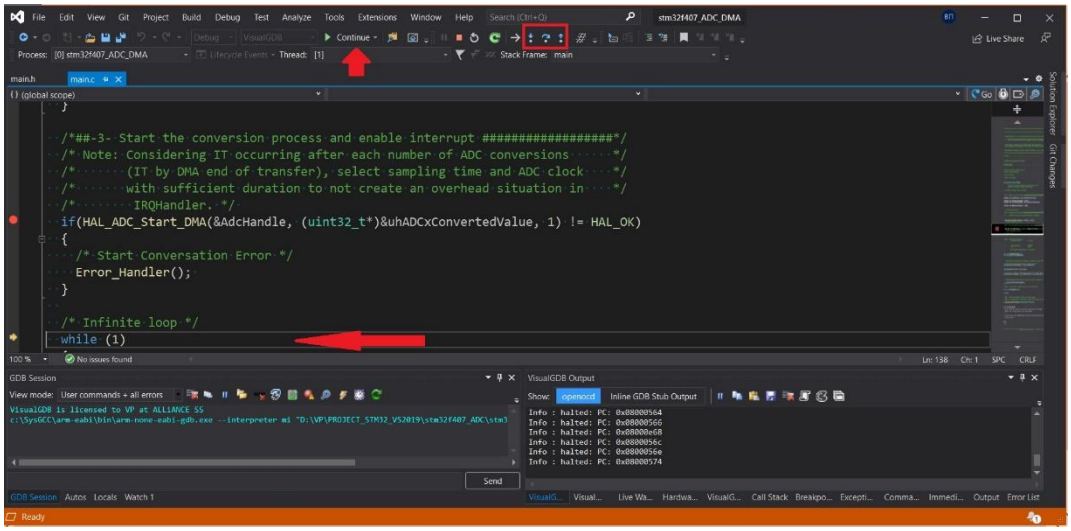


Рис. 1.8в. Запуск режиму компілювання та відлагодження програми, встановлення точки переривання

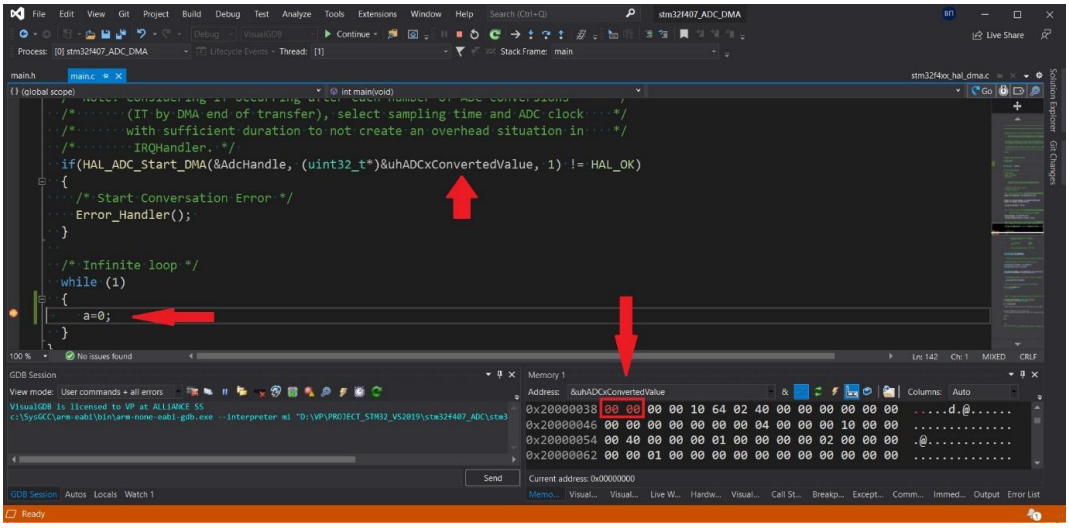


Рис. 1.9а. Читання змінної, що зберігає відлік АЦП

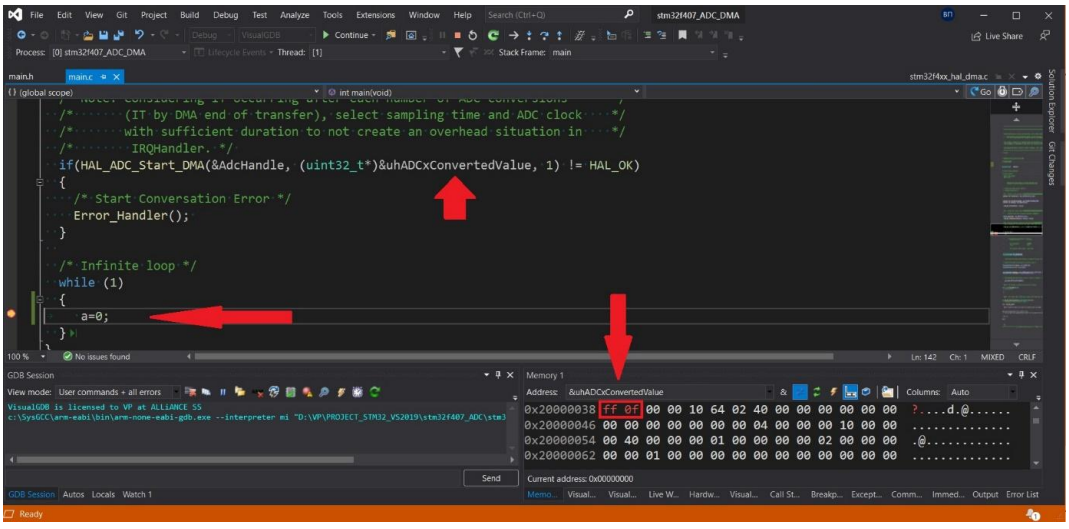
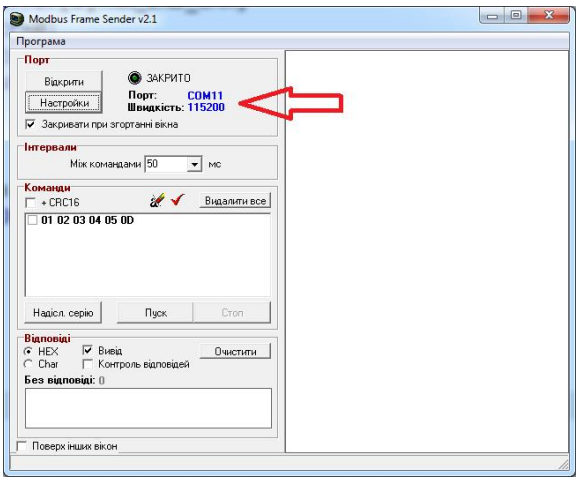


Рис. 1.9б. Читання змінної, що зберігає відлік АЦП

Рис. 1.10. Керування та індикація стенду

Прийом значень АЦП в ПК здійснюється програмою “Modbus FS”.



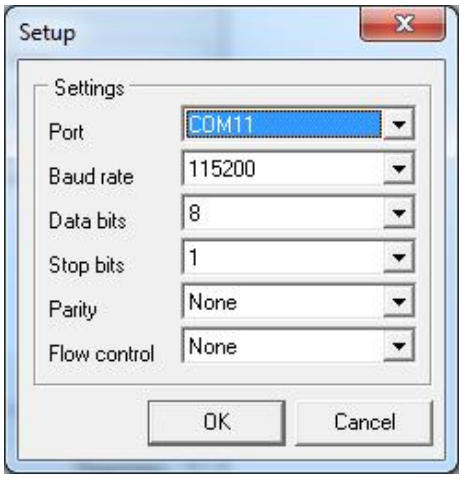


Рис. 1.11. Стартове вікно програми “Modbus FS” *Рис. 1.12. Налаштування програми “Modbus FS*

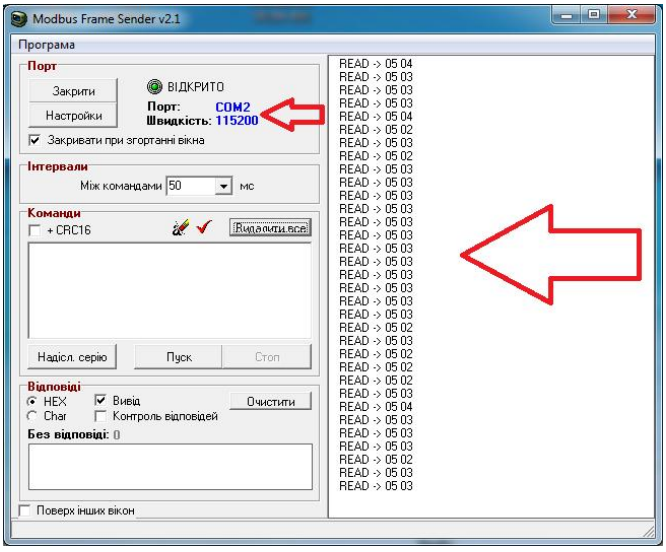


Рис. 1.13. Вивід показів АЦП через UART

МЕТОДИЧНІ МАТЕРІАЛИ

Лабораторний стенд реалізовано на базі відлагоджувального модуля **STM32F4DISCOVERY.**

Модуль STM32F4DISCOVERY

Модуль призначений для відлагодження апаратних вузлів мікропроцесорної ситеми на базі мікроконтролерів STM32F407/417. На платі модуля розміщено мікроконтролер STM32F407VGT6 в корпусі LQFP100 з вузлом синхронізації на базі кварцевого резонатора 8MHz, вузол USB, аудіо-ЦАП, 4 світлодіоди користувача, кнопка RESET, кнопка користувача та контактні групи підключені до виводів мікроконтролера. Крім цього на платі розміщено адаптер ST-LINK/V2 який використовується для відлагодження в реальному часі та програмування пам'яті Flash мікроконтролера. Адаптер може бути відключений від мікроконтролера на платі та використовуватися для відлагодження та програмування плати користувача через інтерфейс SWD.

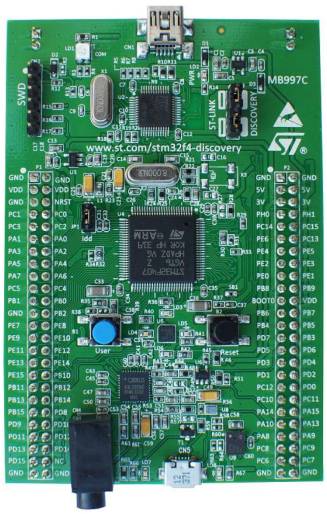


Рис. 1.14. Зовнішній вигляд модуля STM32F4DISCOVERY

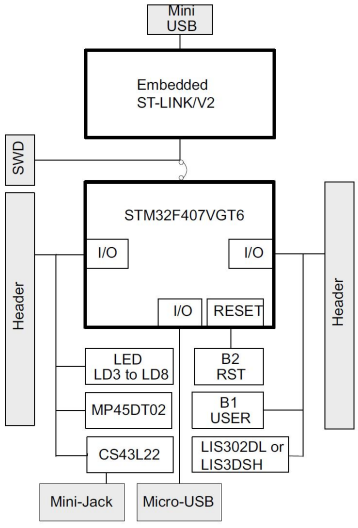


Рис. 1.15. Структура модуля STM32F4DISCOVERY

На платі модуля розміщено контактні групи **Header**, на які виведено відповідні лінії мікроконтролера STM32F407VGT6, див. таблицю **Table 5. MCU pin description versus board function** у файлі *STM32F4DISCOVERY User manual.pdf:*

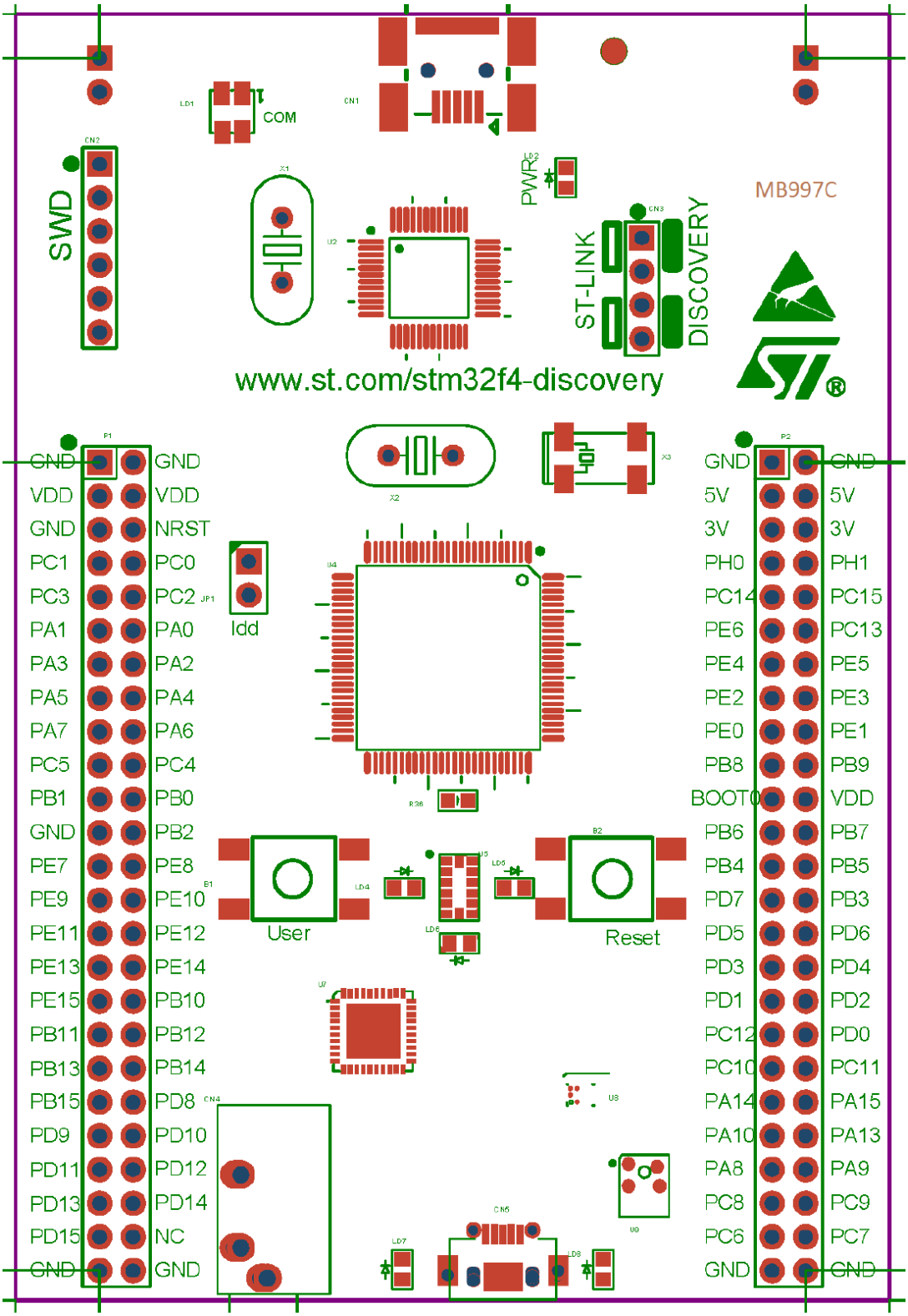


Рис. 1.16. Контакти ***Header*** *модуля STM32F4DISCOVERY*

На платі модуля розміщено відлагоджувальний вузол **ST-LINK/V2**, який забезпечує програмування програмної Flash-пам'яті мікроконтролера та відлагодження в реальному часі записаної програми. Крім цього на платі розміщено **2 кнопки, 4 кольорових світлодіоди LED, Audio DAC, USB, Accelerometer, ST MEMS microphone,** див. *STM32F4DISCOVERY User manual.pdf*. Принципова схема модуля STM32F4DISCOVERY наведена у *STM32F4DISCOVERY User manual.pdf*.

Ядро Cortex-M4F

Розроблено для проєктування вбудованих мікрокомп'ютерних систем середньої продуктивності з низькою споживною потужністю. Архітектурно це 32-розрядне RISC-ядро типу ARM нового покоління з повним набором інструкцій ARM, реалізацією інструкцій DSP, вбудованим процесором плаваючої коми (FPU) та максимальною робочою частотою до 200 MHz. Ядро має вбудований модуль захисту пам'яті який підвищує безпеку роботи виконання програм. Для оптимального програмування FPU використовується спеціальна мета-мова. Все це дозволяє ефективно обробляти сигнали та реалізовувати складні алгоритми керування.

На базі ядра Cortex-M4F різні фірми освоїли випуск широкої номенклатури мікроконтролерів з різними об'ємами інтегрованої на кристалі пам'яті та різноманітною номенклатурою інтегрованих периферійних пристроїв. Одними з таких мікроконтролерів є мікроконтролери серії STM32F4xx фірми STMicroelectronics які стали досить популярними серед розробників.

Вказані мікроконтролери характеризуються наявністю інтегрованої швидкодіючої пам'яті Flash до 1 Mb та наявністю кеш-пам'яті що забезпечує звертання до пам'яті на максимальній частоті процесора без циклів очікування. Система обробки подій в реальному часі не вимагає втручання процесора що забезпечує обробку без затримок. Номенклатура периферійних пристроїв різних мікроконтролерів поряд з класичними інтерфейсними пристроями включає такі комунікаційні інтерфейси як Ethernet MAC з підтримкою стандарту IEEE1588, CAN, USB, інтерфейси для підключення відеокамери та графічних дисплеїв, 12 та 16-розрядні багатоканальні швидкодіючі аналого-цифрові перетворювачі (АЦП), спеціалізовані цифро-аналогові перетворювачі (ЦАП). Для забезпечення високої продуктивності при обміні даними на кристалах інтегровані канали прямого доступу до пам'яті. Деякі мікроконтролери мають інтегровані на кристалі такі специфічні вузли як криптографічний процесор, модуль обчислення контрольних сум CRC, генератор випадкових чисел.

Важливою особливістю вказаних мікроконтролерів є можливість функціонування в режимі пониженого споживання при батарейному живленні. Також при пропаданні основного живлення забезпечується зберігання важливих даних в SRAM з батарейним живленням та функціонування таймера реального часу. Мікроконтролери випускаються в різних корпусах з різною кількістю виводів від 64 до 176, що дозволяє оптимізувати апаратні засоби при різних застосуваннях. Напруга живлення мікроконтролерів від 1.8V до 3.6V , температура в робочому режимі від – 40 до +105 градусів С.

ВНУТРІШНЯ СТРУКТУРА STM32407

Основні характеристики

- ядро 32 розряди;
- максимальна частота ядра 168 MHz;
- вбудований процесор плаваючої коми FPU;
- інструкції ARM та DSP;

• вбудована Flash пам’ять програм до 1Mb та SRAM даних до 192 Kb з можливістю зовнішнього розширення;

• SRAM 4Kb з зовн. резервним живленням;

• широка номенклатура інтегрованої периферії;

• VDD = 3.3 V (1.8 V ≤ VDD ≤ 3.6 V)

Вузли STM32F40x

• ядро ARM® Cortex™-M4F 168 MHz;

• система синхронізації PLL;

• інтегрований процесор FPU;

• інтегрована Flash пам’ять програм до 1Mb;

• інтегрована SRAM даних до 192 Kb;

• інтегрована резервна SRAM даних 4Kb (Vbat);

• адаптивний акселератор памя’ті (ART Accelerator);

• блок захисту пам’яті (MPU);

• модуль обчислення контрольних сумм (CRC unit);

• генератор випадкових чисел (RNG);

• мульти-AHB матриця шин 32 р;

• контролер прямого доступу до пам’яті (DMA);

• контролер статичної пам’яті (FSMC) з можливісю конфігурації керування PKI;

• контролер вкладених векторних переривань (NVIC);

• контролер зовнішніх переривань (EXTI);

• широка номенклатура інтерфейсних вузлів для паралельного та послідовного обміну (GPIO, (SDIO)-SD/SDIO/MMC, USART, SPI, I²C, CAN, USB, Ethernet, I2S);

• інтерфейс цифрової камери (DCMI);

• інтегровані 12р АЦП та ЦАП;

• універсальні таймери, RTC, WDT, SysTick ;

• вбудовані макро-регістри трасування (Embedded Trace Macrocell, ETM);

• JTAG та SW інтерфейси для трасування та програмування інтегрованої Flash;

• інтегрований стабілізатор напруги живлення;

• вузол керування живленням;

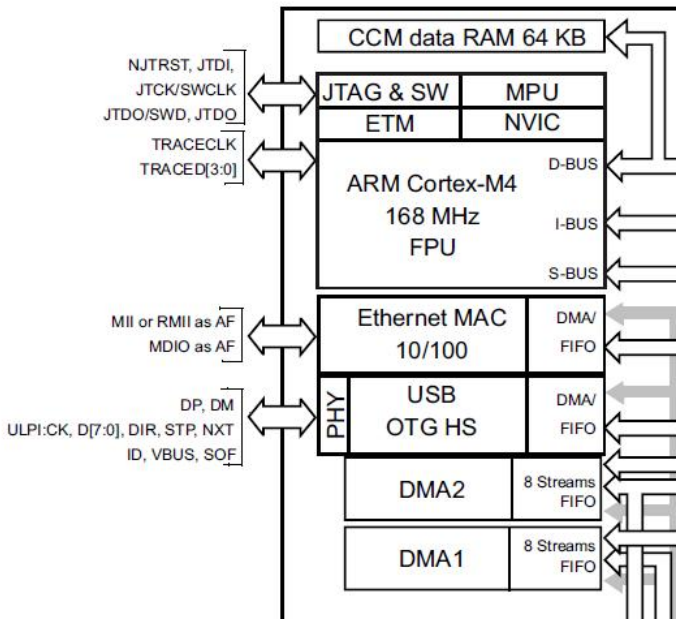


Рис. 1.17а. Внутрішня структура STM32F407

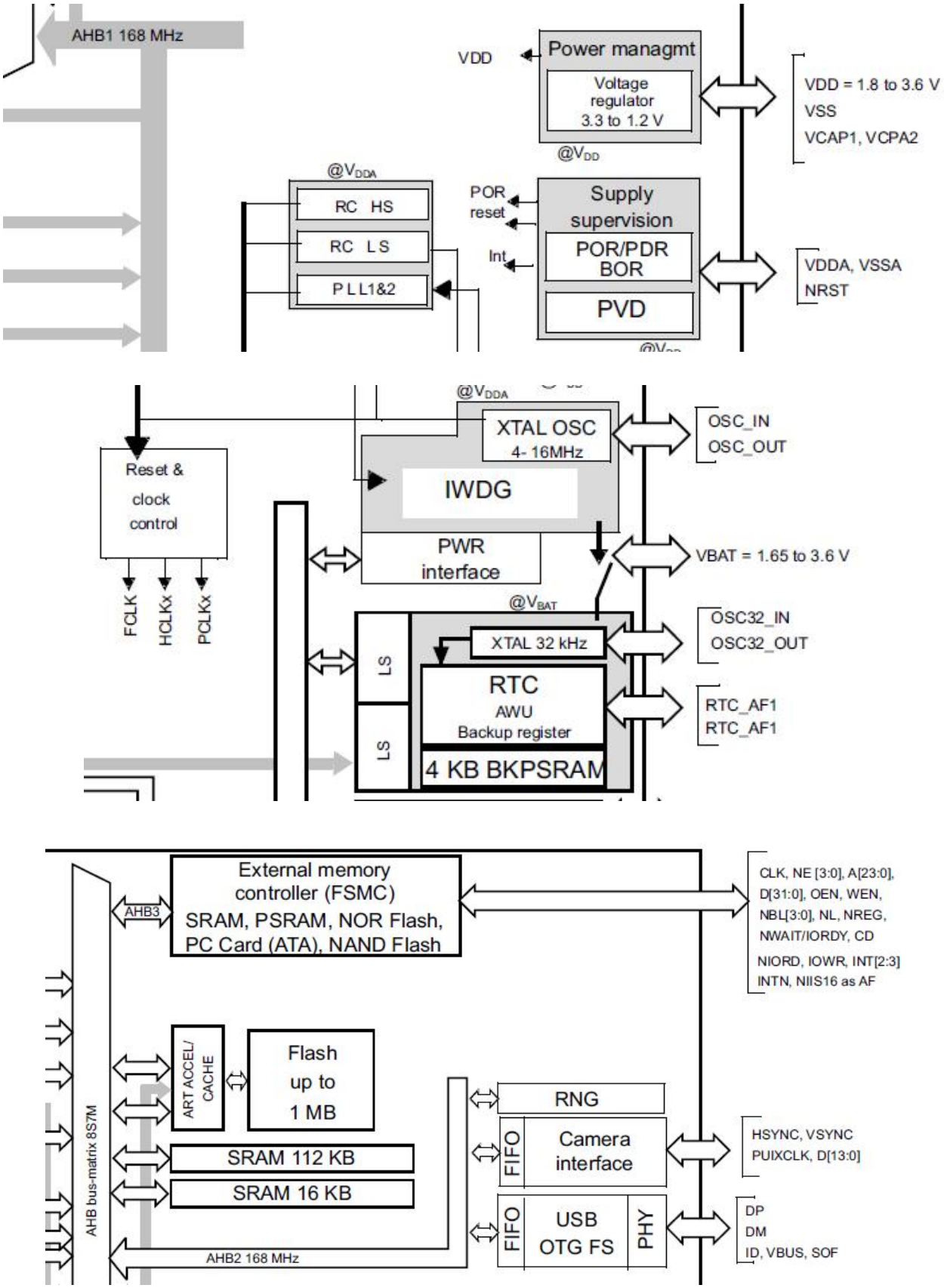


Рис. 1.17б. Внутрішня структура STM32F407

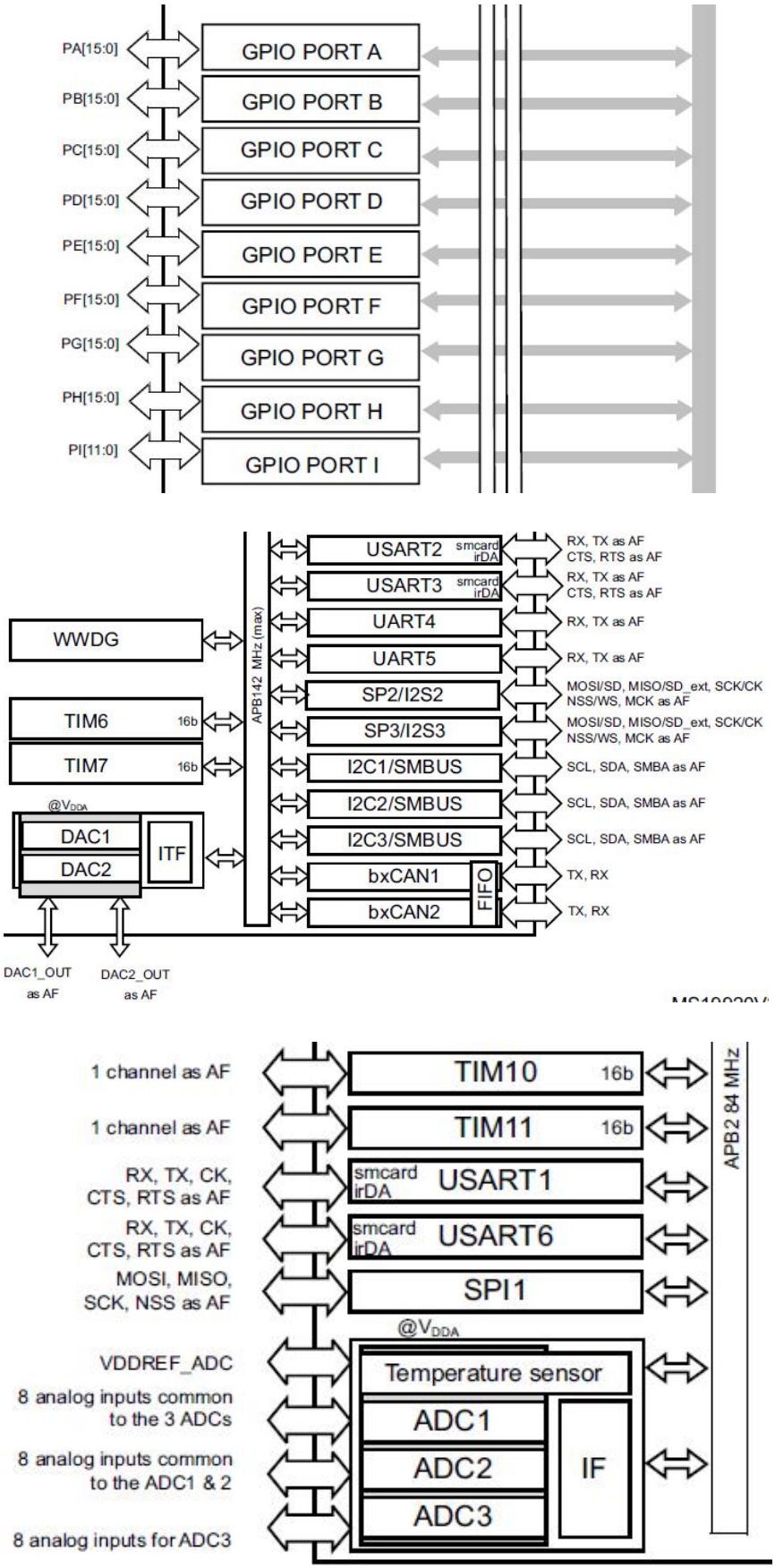


Рис. 1.17в. Внутрішня структура STM32F407

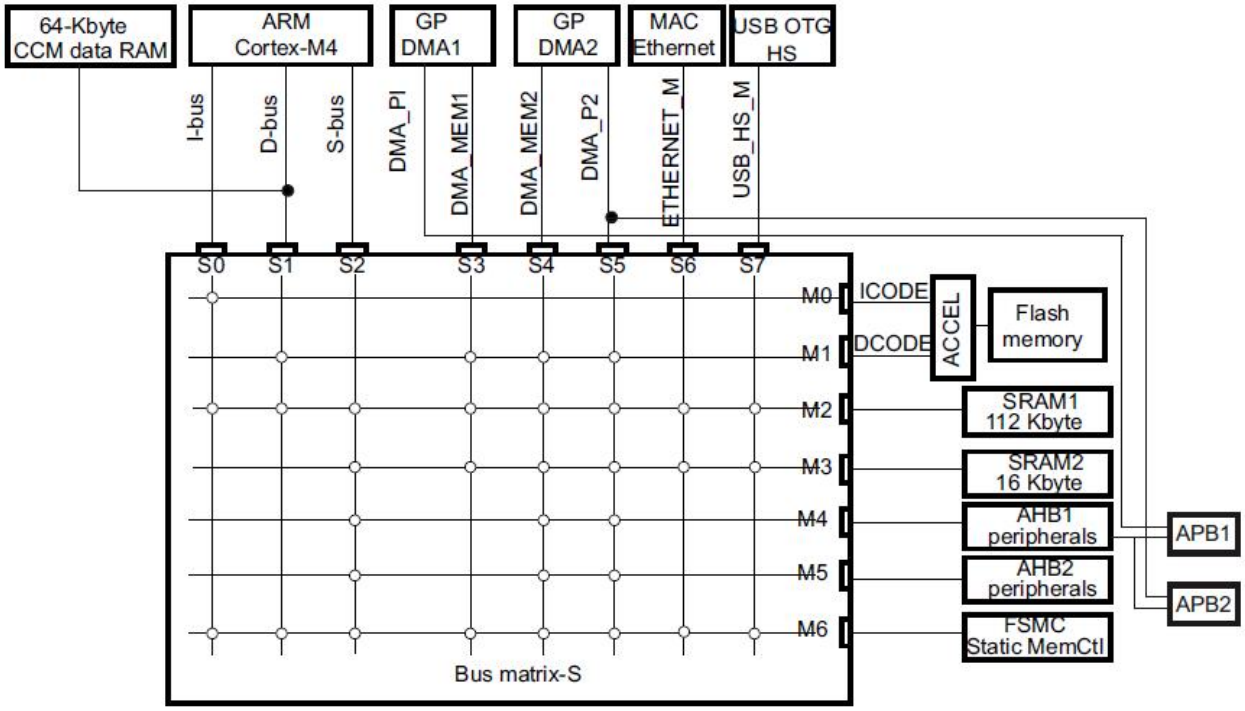


Рис. 1.18. Мульти-AHB матриця шин 32 р

КПДП (DMA)
memory-to-memory, peripheral-to-memory and memory-to-peripheral

- SPI and I2S
- I2C
- USART
- General-purpose, basic and advanced-control timers TIMx
- DAC
- SDIO
- Camera interface (DCMI)
- ADC.

Живлення

• VDD = 1.8 to 3.6 V: external power supply for I/Os and the internal regulator (when enabled), provided externally through VDD pins.

• VSSA, VDDA = 1.8 to 3.6 V: external analog power supplies for ADC, DAC, Reset blocks, RCs and PLL. VDDA and VSSA must be connected to VDD and VSS, respectively.

• VBAT = 1.65 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when VDD is not present.

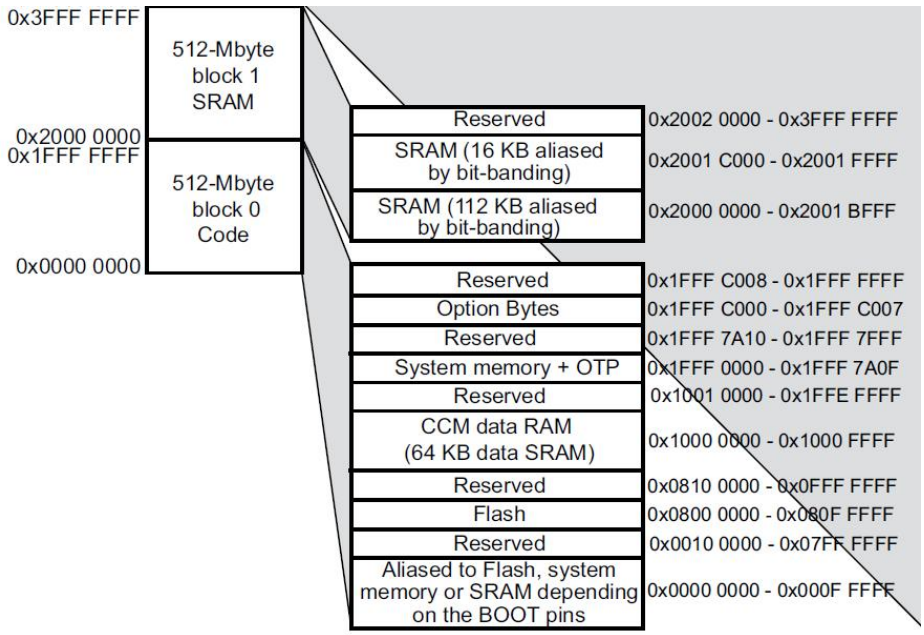


Рис. 1.19а. ***Структура пам'яті STM32F40x***

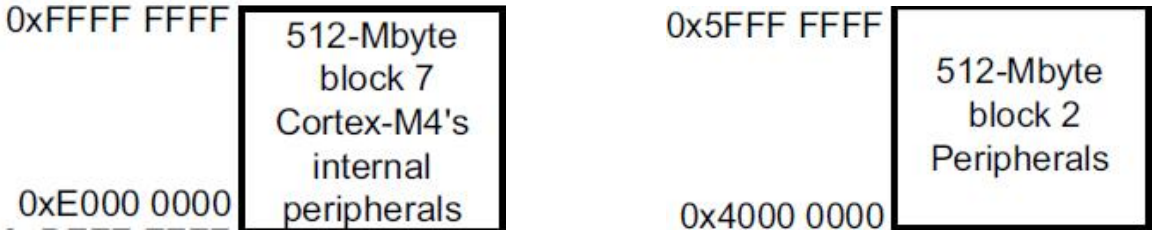


Рис. 1.19б. Структура пам'яті STM32F40x

Bus	Boundary address	Peripheral
	0xE00F FFFF - 0xFFFF FFFF	Reserved
Cortex-M4	0xE000 0000 - 0xE00F FFFF	Cortex-M4 internal peripherals
	0xA000 1000 - 0xDFFF FFFF	Reserved
AHB3	0xA000 0000 - 0xA000 0FFF	FSMC control register
	0x9000 0000 - 0x9FFF FFFF	FSMC bank 4
	0x8000 0000 - 0x8FFF FFFF	FSMC bank 3
	0x7000 0000 - 0x7FFF FFFF	FSMC bank 2
	0x6000 0000 - 0x6FFF FFFF	FSMC bank 1
	0x5006 0C00- 0x5FFF FFFF	Reserved
AHB2	0x5006 0800 - 0x5006 0BFF	RNG
	0x5005 0400 - 0x5006 07FF	Reserved
	0x5005 0000 - 0x5005 03FF	DCMI
	0x5004 0000- 0x5004 FFFF	Reserved
	0x5000 0000 - 0x5003 FFFF	USB OTG FS
	0x4008 0000- 0x4FFF FFFF	Reserved

Рис. 1.20а. Регістри STM32F40x

Bus	Boundary address	Peripheral
AHB1	0x4004 0000 - 0x4007 FFFF	USB OTG HS
	0x4002 9400 - 0x4003 FFFF	Reserved
	0x4002 9000 - 0x4002 93FF	ETHERNET MAC
	0x4002 8C00 - 0x4002 8FFF	
	0x4002 8800 - 0x4002 8BFF	
	0x4002 8400 - 0x4002 87FF	
	0x4002 8000 - 0x4002 83FF	
	0x4002 6800 - 0x4002 7FFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 5FFF	Reserved
	0x4002 4000 - 0x4002 4FFF	BKPSRAM
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2400 - 0x4002 2FFF	Reserved
	0x4002 2000 - 0x4002 23FF	GPIOI
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA
	0x4001 5800- 0x4001 FFFF	Reserved

Рис. 1.20б. Регістри STM32F40x

АЦП мікроконтролера STM32F407

STM32F407 має в своєму складі три 12-бітних аналого-цифрових перетворювачі (**АЦП**), кожний з яких може отримувати вхідну напругу через спеціально призначені (мультиплексовані) входи і перетворювати їх в числовий код. Вхідна напруга порівнюється з опорною напругою **Reference Voltage (VREF)**. Опорна напруга може братися або з аналогового живлення **VDDA** або з інтегрованого чи зовнішнього джерела опорної напруги **VREF**.

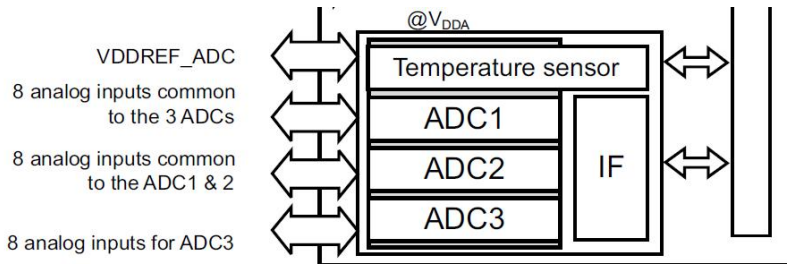


Рис. 1.21. АЦП мікроконтролера STM32F407

Крім зовнішніх входів доступні також кілька внутрішніх каналів: канал внутрішнього температурного сенсора (Vsense), канал для внутрішньої опорної напруги (Vrefint), канал

для моніторингу зовнішньої напруги живлення батареї (Vbat). Перетворення різних каналів може виконуватися в одноканальному (**single**), багатоканальному (**scan**), безперервному (**continuous**) або переривчастому (**discontinuous**) режимах. Результат роботи АЦП записується в 16-розрядний регістр даних з лівим або правим вирівнюванням. Крім того, АЦП також може працювати в якості аналогового сторожового таймера, який дозволяє програмі визначати момент, коли вхідна напруга виходить за рамки заданого користувачем діапазону і, якщо це відбувається, генерується переривання.

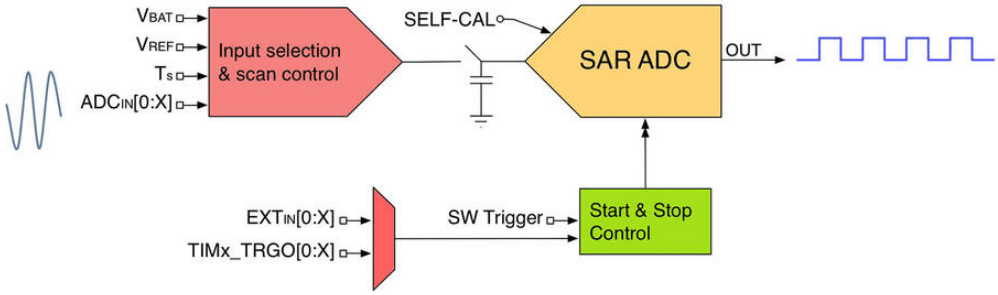


Рис. 1.22. Структура АЦП

Блок вибору каналу і контролю сканування (**Input Selection & Scan Control**) виконує вибір джерела вхідної напруги для АЦП. Залежно від режиму перетворення (single, scan або continuous), цей блок автоматично перемикає вхідні канали, так що кожен з них може бути зчитаний з певним періодом. Вихід даного блоку підключений до АЦП (**SAR ADC**). На Рис. 2 зображено блок контролю запуску і зупинки (**Start&Stop&Control**). Його роль полягає в контролі процесу перетворення АЦП, він може перемикатися як програмно, так і по подіях від деяких вхідних джерел. Крім того, внутрішньо він підключений до лінії TRGO декількох таймерів, тому може бути реалізовано перетворення за таймером в режимі DMA.

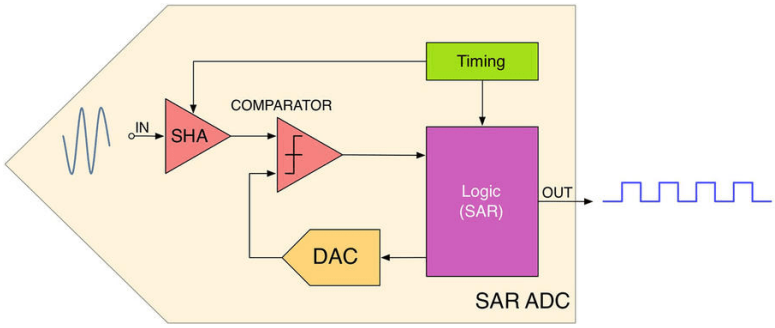


Рис. 1.23. Внутрішня структура АЦП

На Рис. 3 показано основні вузли: вузол вибірки і зберігання (**Sample-and-Hold SHA**), компаратор (**Comparator**), цифро-аналоговий перетворювач (ЦАП), (**DAC**) та вузол логіки (**Logic SAR**), який обчислює цифрове значення коду по алгоритму послідовного наближення. Алгоритм послідовного наближення обчислює напругу вхідного сигналу шляхом її порівняння з величиною, що генерується від внутрішнього ЦАП (DAC): якщо вхідний сигнал більше, ніж ця внутрішня опорна напруга, то відбувається подальше збільшення цієї опорного напруги поки вхідний сигнал не стане менше за неї. Остаточний результат

співвідноситься з числовим представленням в межах від нуля до максимального значення 12 бітного цілого беззнакового числа, тобто 2 в 12 степені – 1 = 4095. Якщо прийняти що VREF = 3300мВ то 3300мВ представляється числом 4095. Це означає що 1 одиниця АЦП рівна 3300/4095 = 0.8мВ. Нехай вхідний сигнал дорівнює 2.5В, тоді він буде перетворений в цифровий код **3102** в десятковому представленні:

$$x = \frac{4095}{3300\text{мВ}} \times 2500\text{мВ} = 3102$$

Режими перетворення

АЦП має кілька режимів роботи, які можуть використовуватися в різних сценаріях роботи програми

Один канал, одноразовий режим перетворення

Це найпростіший режим роботи АЦП. У цьому режимі, АЦП здійснює одне перетворення (одна вибірка) одного каналу, як показано на рис. 4, і завершує свою роботу, коли перетворення завершено.

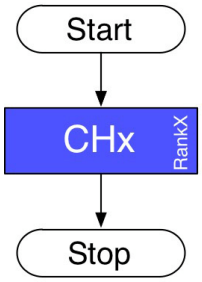


Рис. 1.24. Один канал, одне перетворення

Декілька каналів, одноразовий режим перетворення (багатоканальний режим)

Цей режим використовується для отримання по одній вибірці з кожного каналу в незалежному режимі. Можна задати будь-яку послідовність каналів АЦП з різним часом перетворення і різної черговості, але ви не можете зупинити АЦП під час процесу перетворення для того, щоб змінити час перетворення наступного каналу. Цей режим додатково навантажує процесорне ядро, складний програмно і його краще здійснювати в режимі канал прямого доступу (DMA).

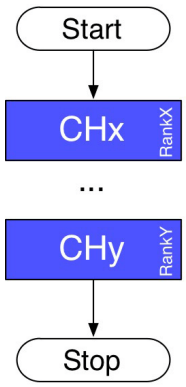


Рис. 1.25. Багатоканальний режим

Один канал, режим безперервного перетворення

Цей режим дозволяє перетворювати один канал безперервно і нескінченно довго в режимі перетворення регулярних каналів. Безперервний режим роботи дозволяє АЦП працювати фоново. АЦП перетворює значення одного каналу без будь-якого втручання з боку процесора. В циклічному режимі може бути використаний DMA, що зменшує навантаження на процесорне ядро.

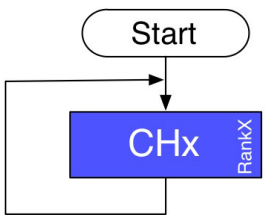


Рис. 1.26. Режим циклічного перетворення одного каналу

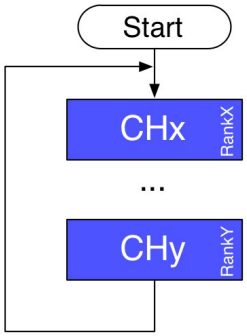


Рис. 1.27. Режим циклічного перетворення декількох каналів

Режим циклічного перетворення декількох каналів

Цей режим також називають багатоканальним безперервним режимом і він може використовуватися для перетворення декількох каналів в незалежному режимі. Використовуючи індекси каналів **Rank**, доступна настройка будь-якj] послідовності каналів з різним часом вибірок і черговістю перетворення. Цей режим схожий на простий багатоканальний режим з тією лише різницею, що перетворення не зупиняється після перетворення останнього каналу в послідовності регулярних каналів, а починається по новій з самого першого каналу і триває цей цикл нескінченно довго. У цьому режимі також доступний DMA.

Режим перетворення інжектованих каналів

Цей режим призначений для використання в режимі запуску перетворення по зовнішній події або програмно. Група інжектованих каналів має більш високий пріоритет ніж група регулярних каналів. Її перетворення може перервати перетворення поточного каналу в групі регулярних каналів.

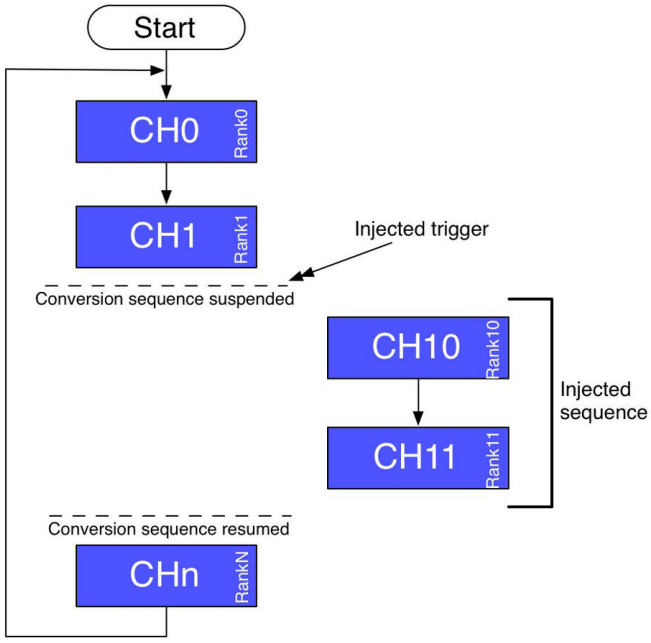


Рис. 1.28. Режим перетворення інжектованих каналів

Вибір каналу

Вхідні канали фіксовані і прив'язані до конкретних виводів мікроконтролера (IN0, IN1 і т. д.). В той же час вони можуть бути логічно змінені на будь-яку черговість вибірок. Перепризначення каналів реалізовано шляхом призначення кожному каналу індексу в діапазоні від 1 до 16. Цей індекс називається Rank в CubeHAL і відповідає черговості перетворення каналу в послідовності регулярних каналів.

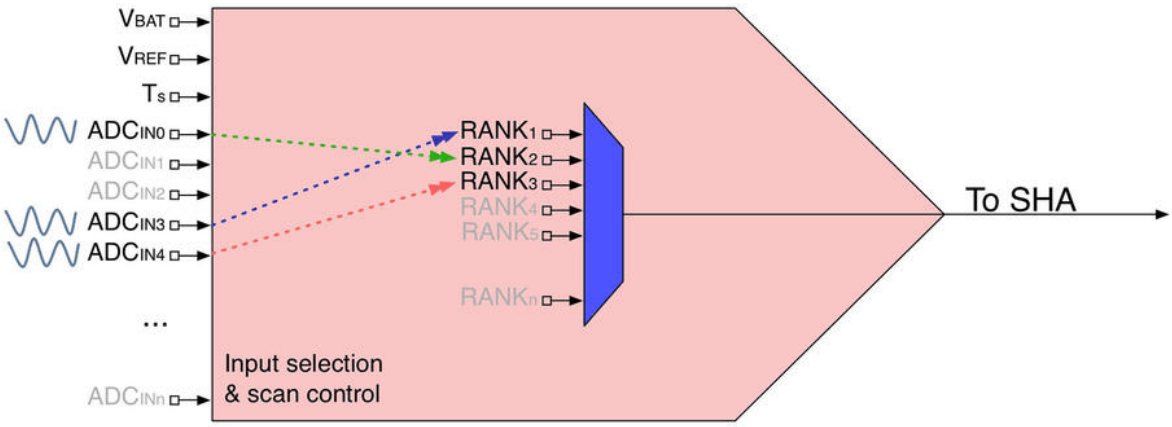


Рис. 1.29. Переназначення каналів через індекс RANK

Конфігурація канал / індекс представлена структурою С **ADC_ChannelConfTypeDef**, яка визначена таким способом:

```
typedef struct {
uint32_t Channel;          /* Канал, якому назначається індекс Rank */
uint32_t Rank;             /* Індекс черговості каналу */
uint32_t SamplingTime;     /* Час одної вибірки для вибраного каналу */
uint32_t Offset;           /* Зарезервовано, может бути встановлено в 0 */
} ADC_ChannelConfTypeDef;
```

- **Channel**: встановлює ідентифікатор каналу; може приймати значення **ADC_CHANNEL_0, ADC_CHANNEL_1 … ADC_CHANNEL_N** в залежності від доступного числа каналів.
- **Rank**: відповідає індексу черговості для конкретного каналу.
- **SamplingTime**: встановлює час одної вибірки для каналу, визначається числом тактів АЦП та вибирается з списку конкретних значень.

```
/*##-2- Configure ADC regular channel ######################################*/
/* Note: Considering IT occurring after each number of size of              */
/*       "uhADCxConvertedValue" ADC conversions (IT by DMA end              */
/*       of transfer), select sampling time and ADC clock with sufficient   */
/*       duration to not create an overhead situation in IRQHandler.        */
sConfig.Channel = ADCx_CHANNEL;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
sConfig.Offset = 0;

if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
{
  /* Channel Configuration Error */
  Error_Handler();
}
```

Рис. 1.30. Конфігурація каналів АЦП

```
/* Definition for ADCx Channel Pin */
#define ADCx_CHANNEL_PIN                GPIO_PIN_0
#define ADCx_CHANNEL_GPIO_PORT          GPIOB

/* Definition for ADCx's Channel */
#define ADCx_CHANNEL                    ADC_CHANNEL_8

/* Definition for ADCx's DMA */
#define ADCx_DMA_CHANNEL                DMA_CHANNEL_0
#define ADCx_DMA_STREAM                 DMA2_Stream0

/* Definition for ADCx's NVIC */
#define ADCx_DMA_IRQn                   DMA2_Stream0_IRQn
#define ADCx_DMA_IRQHandler             DMA2_Stream0_IRQHandler
```

Рис. 1.31. Параметри для налаштування АЦП

PB0	TIM3_CH3/ TIM8_CH2N/ OTG_HS_ULPI_D1/ ETH_MII_RXD2/ TIM1_CH2N/ ADC12_IN8

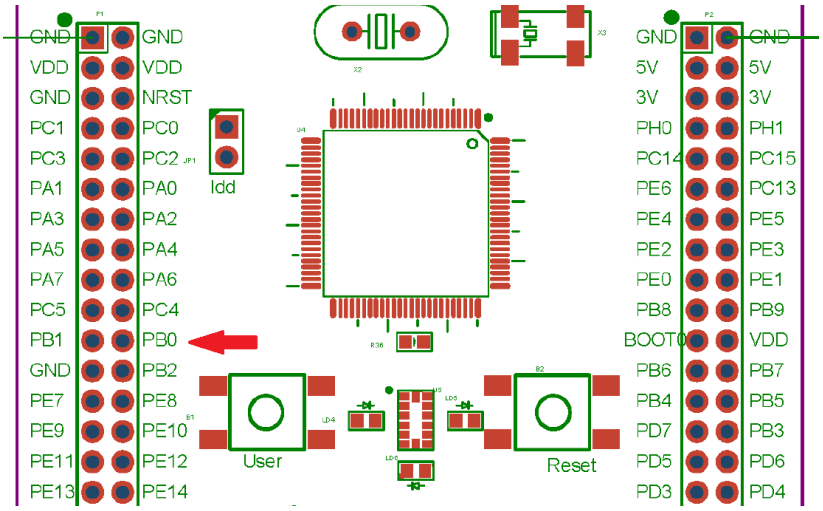


*Рис. 1.32. Контакт **PB0** каналу **IN8** АЦП*

Багаторазове неперервне перетворення в режимі DMA

Для виконання багаторазового перетворення в режимі DMA:

- встановити параметр **hadc.Init.DMAContinuousRequests** в **ENABLE**.
- визвати **HAL_ADC_Start_DMA()** для запуску перетворення.

Якщо замість цього встановити **hadc.Init.DMAContinuousRequests** в **DISABLE**, то необхідно буде кожен раз викликати **HAL_ADC_Stop_DMA ()** в кінці кожної послідовності перетворень, а після знову викликати **HAL_ADC_Start_DMA ()**. В іншому випадку перетворення не буде виконано.

```
/*##-1- Configure the ADC peripheral #######################################*/
AdcHandle.Instance = ADCx;

AdcHandle.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
AdcHandle.Init.Resolution = ADC_RESOLUTION_12B;
AdcHandle.Init.ScanConvMode = DISABLE;
AdcHandle.Init.ContinuousConvMode = ENABLE;
AdcHandle.Init.DiscontinuousConvMode = DISABLE;
AdcHandle.Init.NbrOfDiscConversion = 0;
AdcHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
AdcHandle.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_CC1;
AdcHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;
AdcHandle.Init.NbrOfConversion = 1;
AdcHandle.Init.DMAContinuousRequests = ENABLE;
AdcHandle.Init.EOCSelection = DISABLE;

if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
{
  /* Initialization Error */
  Error_Handler();
}
```

Рис. 1.33. Конфігурація режимів та параметрів АЦП

```
/*##-3- Start the conversion process and enable interrupt ##################*/
/* Note: Considering IT occurring after each number of ADC conversions      */
/*       (IT by DMA end of transfer), select sampling time and ADC clock    */
/*       with sufficient duration to not create an overhead situation in    */
/*       IRQHandler. */
if(HAL_ADC_Start_DMA(&AdcHandle, (uint32_t*)&uhADCxConvertedValue, 1) != HAL_OK)
{
  /* Start Conversation Error */
  Error_Handler();
}
```

Рис. 1.34. Запуск перетворення АЦП в режимі DMA

Вузол відображення на базі алфавітно-цифрового LCD модуля BC1602A

В процесі виконання програми перетворення АЦП результат перетворення виводиться на символьний індикатор BC1602A та передається в комп’ютер через порт UART.

Рис. 1.35. Алфавітно-цифровий LCD модуль BC1602A фірми Bolymin

Таблиця 1.1. Контакти LCD модуля BC1602A

Pin	Symbol	Function
1	Vss	GND
2	Vdd	+5V(+3V option)
3	Vo	Contrast adjustment
4	RS	H/L register select signal
5	R/W	H/L read/write signal
6	E	H --> L enable signal
7~14	DB0~7	H/L data bus line
15	A/Vee	Power supply for B/L (+)
16	K	Power supply for B/L (GND)

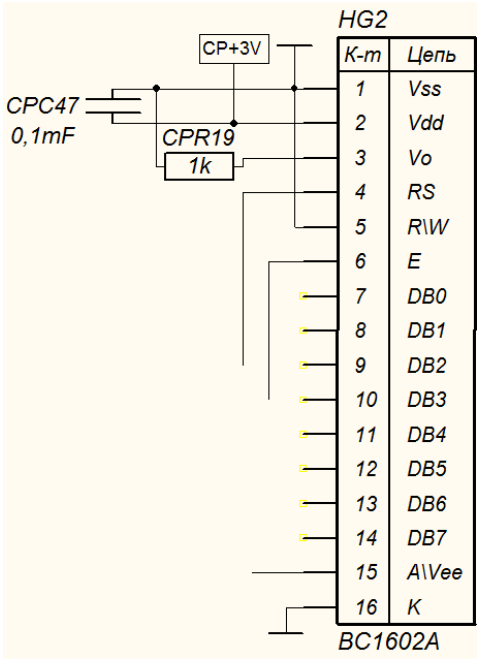


Рис. 1.36. Фрагмент вузла принципової схеми з BC1602A

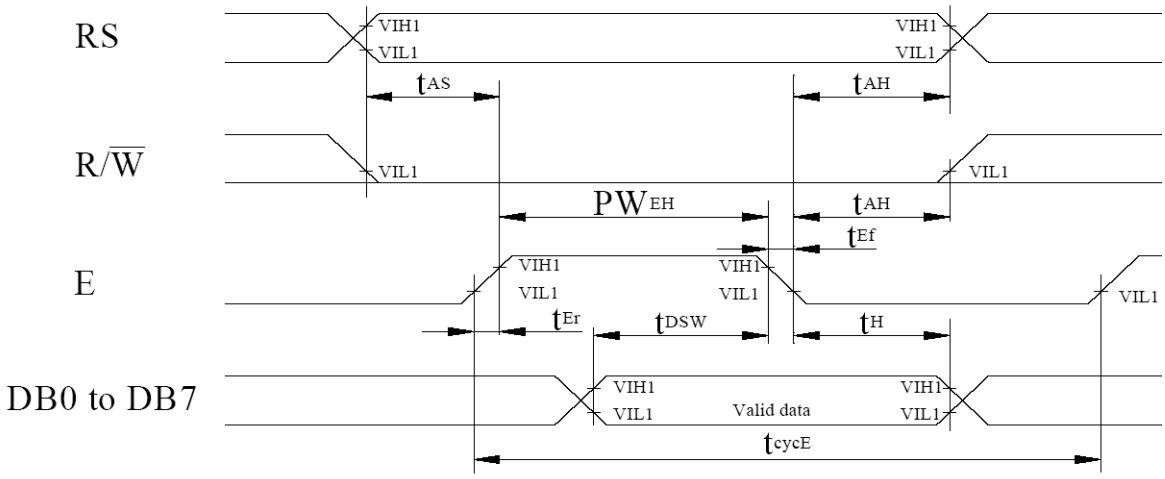


Рис. 1.37. Часова діаграма запису символа в індикатор

Таблиця 1.2. Таблиця кодів символів BC1602A

BC1602A – процедура ініціалізації

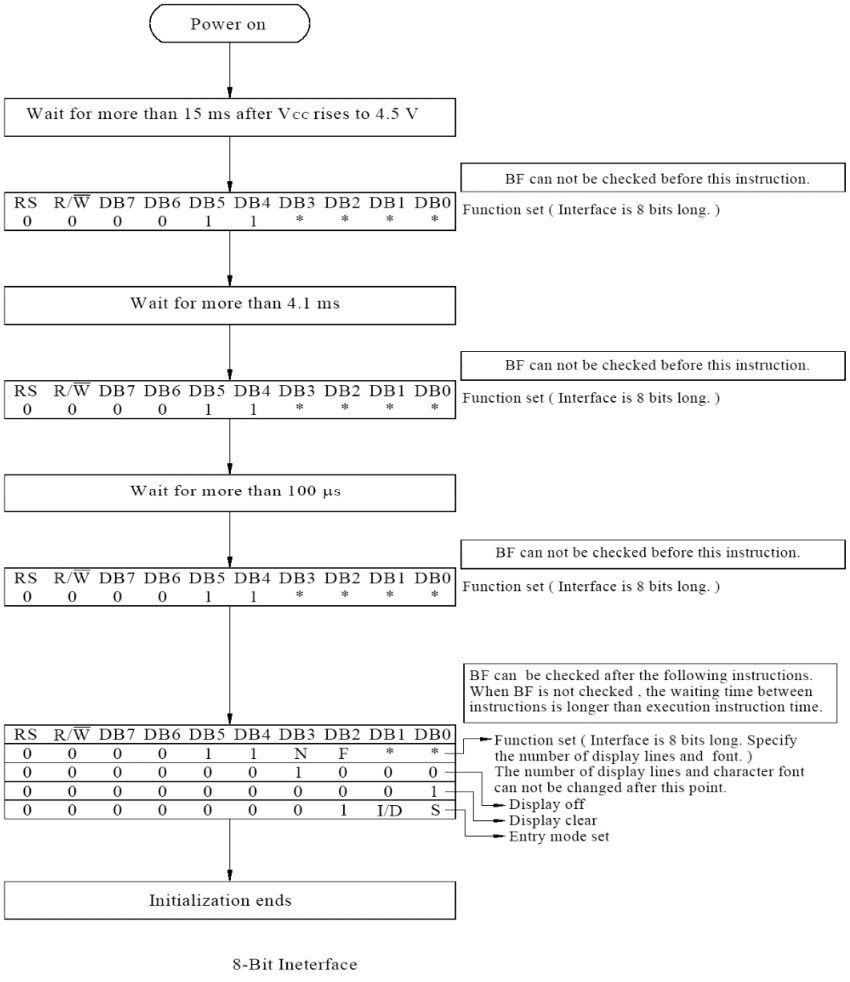


Рис. 1.38. Процедура ініціалізації BC1602A

Керування режимами індикатора ВС1602А здійснюється за допомогою інтегрованого на платі індикатора контролера типу HD44780, див. файл *hd44780u.pdf*.

Символьний індикатор ВС1602А підключено через лінії паралельного порту GPIO PORT B (PB) в режимі 4-х ліній даних, див. **Table 5. MCU pin description versus board function (continued)** документу *STM32F4DISCOVERY User manual.pdf*


```
//LCD1602
#define     LCM_OUT              GPIOB->ODR

#define     LCM_PIN_RS           GPIO_PIN_8          // PB8
#define     LCM_PIN_EN           GPIO_PIN_1          // PB1
#define     LCM_PIN_D7           GPIO_PIN_7          // PB7
#define     LCM_PIN_D6           GPIO_PIN_6          // PB6
#define     LCM_PIN_D5           GPIO_PIN_5          // PB5
#define     LCM_PIN_D4           GPIO_PIN_4          // PB4
```

Рис. 1.39. Підключення ВС1602А через лінії порту PB

```
__GPIOB_CLK_ENABLE();

GPIO_InitTypeDef GPIO_InitStructure;


GPIO_InitStructure.Pin   = GPIO_PIN_1 | GPIO_PIN_4 | GPIO_PIN_5 |
                           GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8;
GPIO_InitStructure.Mode  = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
GPIO_InitStructure.Pull  = GPIO_NOPULL;
HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Рис. 1.40. Ініціалізація ліній порту PB

Регістри паралельних портів

GPIO Registers

Configuration Registers

STM32 містить чотири регістри конфігурації для кожного з портів:

- Port mode register – GPIOx_MODER
- Output type register – GPIOx_OTYPER
- Speed register – GPIOx_OSPEEDR
- Pull-up/Pull-down register – GPIOx_PUPDR

Кожен з цих регістрів має довжину 32 біти, хоча і не всі біти використовуються у всіх регістрах.

Port Mode Register – GPIOx_MODER

Цей 32-розрядний регістр має 2 бітні значення, які визначають режим роботи.
Можна встановити такі режими:

Bit Values	Description
00	Input
01	General purpose output
10	Alternate function
11	Analog

Port	Reset Value
A	0xA800 0000
B	0x0000 0280
All others	0x0000 0000

Output Type Register – GPIOx_OTYPER

Bit Value	Description
0	Push/pull output
1	Open drain output

Speed Register – GPIOx_OSPEEDR

Bit Values	Description
00	2 MHz Low speed
01	25 MHz Medium speed
10	50 MHz Fast speed
11	100 MHz High speed on 30pF 80 MHz High speed on 15 pF

Pull-up/Pull-down register – GPIOx_PUPDR

Bit Values	Description
00	No pull-up or pull-down resistor
01	Pull-up resistor
10	Pull-down resistor
11	Reserved

Input data register – GPIOx_IDR

Bit Value	Description
0	High logic value
1	Low logic value

Output data register – GPIOx_ODR

Bit Value	Description
0	High logic value
1	Low logic value

Вузол UART

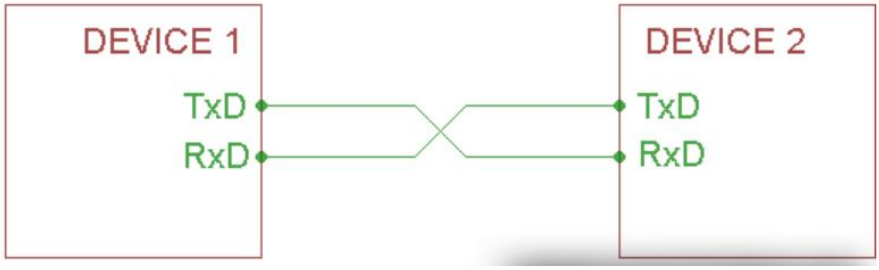
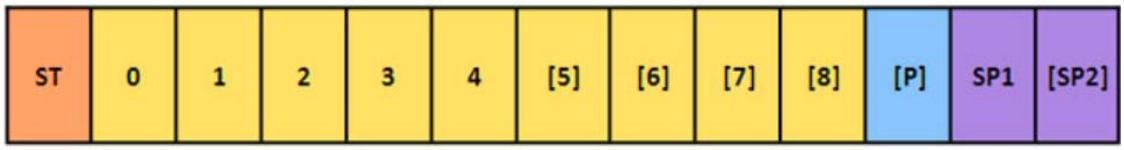


Рис. 1.41. Обмін даними між пристроями через порти UART



- ST – стартовий біт
- 0-8 – біти даних
- P – біт парності
- SP1,SP2 – стопові біти

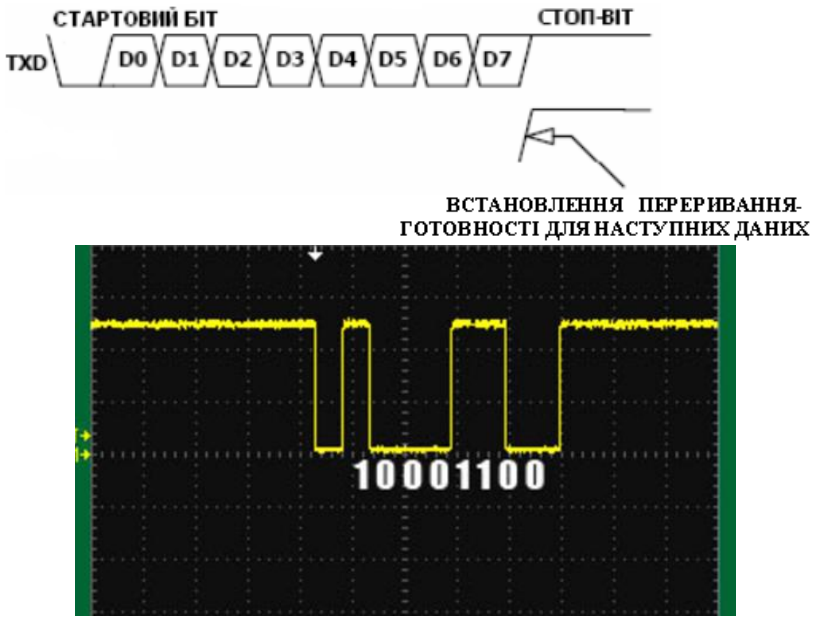


Рис. 1.42. Передача байту через порт UART

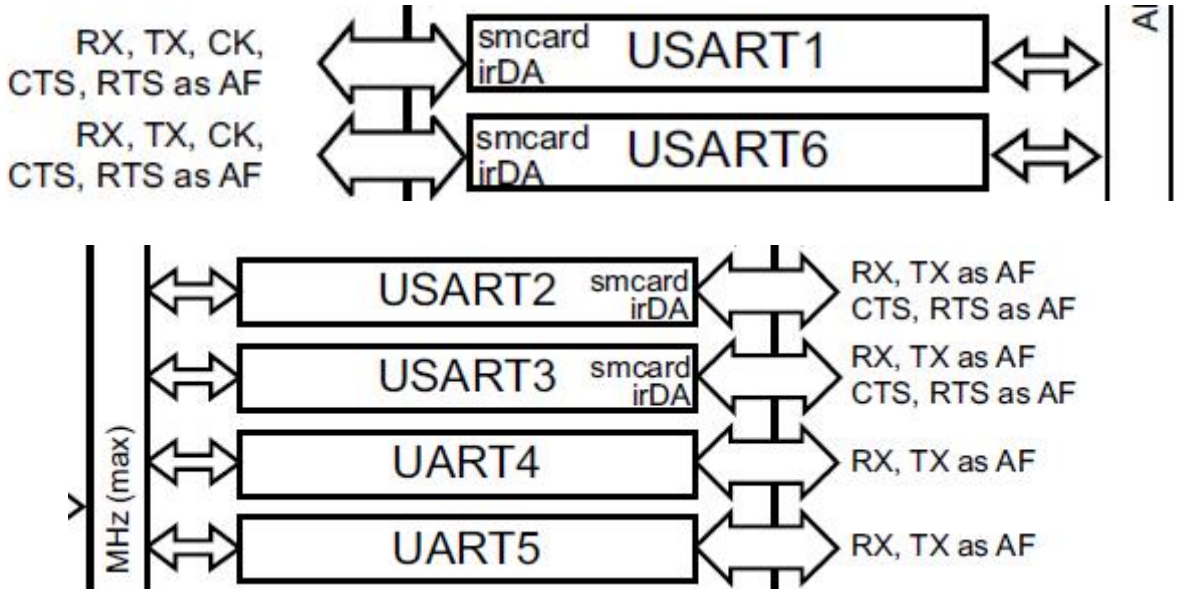


Рис. 1.43. Порти UART мікроконтролера STM32F407VG

Порти USART працюють в режимах синхронної та асинхронної передачі, порти UART – тільки в режимах асинхронної передачі, див. таблицю **Table 147. USART mode configuration** у файлі *RM0090 STM32F405xx_07xx Reference manual.pdf*.

Таблиця 1.3. Режими конфігурації портів UART

USART mode configuration

Table 147. USART mode configuration(1)

USART modes	USART 1	USART 2	USART 3	UART4	UART5	USART 6
Asynchronous mode	X	X	X	X	X	X
Hardware flow control	X	X	X	NA	NA	X
Multibuffer communication (DMA)	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous	X	X	X	NA	NA	X
Smartcard	X	X	X	NA	NA	X
Half-duplex (single-wire mode)	X	X	X	X	X	X
IrDA	X	X	X	X	X	X
LIN	X	X	X	X	X	X

1. X = supported; NA = not applicable.

Рис. 1.44. Режими конфігурації портів UART мікроконтролера STM32F407VG

30.6 USART registers

Refer to *Section 1.1 on page 57* for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by half-words (16 bits) or words (32 bits).

30.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

30.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

30.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рис. 1.45. Регістри портів UART мікроконтролера STM32F407VG

Детальніше про регістри портів UART та їх структуру див. у файлі *RM0090 STM32F405xx_07xx Reference manual.pdf*.

В лабораторній роботі для передачі значень відліків АЦП в комп'ютер використовується порт USART2. Ініціалізація порту наведена на рис. 44.

```
/* Definition for USARTx clock resources */
#define USARTx                           USART2
#define USARTx_CLK_ENABLE()              __HAL_RCC_USART2_CLK_ENABLE();
#define USARTx_RX_GPIO_CLK_ENABLE()      __HAL_RCC_GPIOA_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE()      __HAL_RCC_GPIOA_CLK_ENABLE()

#define USARTx_FORCE_RESET()             __HAL_RCC_USART2_FORCE_RESET()
#define USARTx_RELEASE_RESET()           __HAL_RCC_USART2_RELEASE_RESET()

/* Definition for USARTx Pins */
#define USARTx_TX_PIN                    GPIO_PIN_2
#define USARTx_TX_GPIO_PORT              GPIOA
#define USARTx_TX_AF                     GPIO_AF7_USART2
#define USARTx_RX_PIN                    GPIO_PIN_3
#define USARTx_RX_GPIO_PORT              GPIOA
#define USARTx_RX_AF                     GPIO_AF7_USART2
```

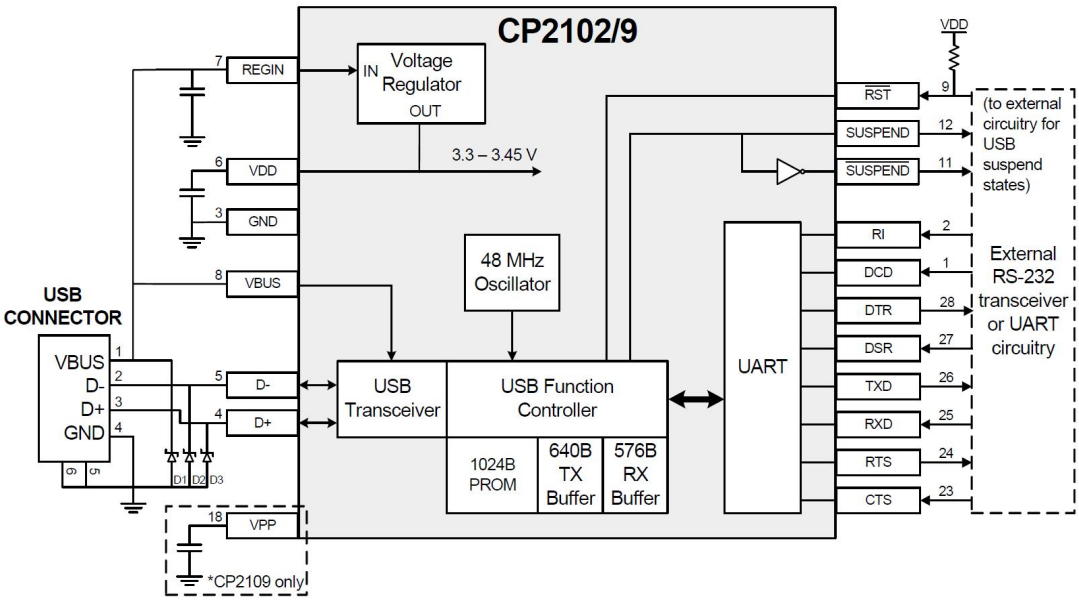
```
UartHandle.Instance        = USARTx;

UartHandle.Init.BaudRate   = 9600;
UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
UartHandle.Init.StopBits   = UART_STOPBITS_1;
UartHandle.Init.Parity     = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
UartHandle.Init.Mode       = UART_MODE_TX_RX;
UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

if(HAL_UART_Init(&UartHandle) != HAL_OK)
{
  Error_Handler();
}
```

Рис. 1.46. Ініціалізація порту USART2

```
if (HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000) != HAL_OK)
{
    Error_Handler();
}
```

Рис. 1.47. Передача байту через порт USART2

Порт UART (USART2) підключається до USB комп’ютера через перетворювач UART-USB типу CP2102, див. файл *CP2102-9.pdf*.



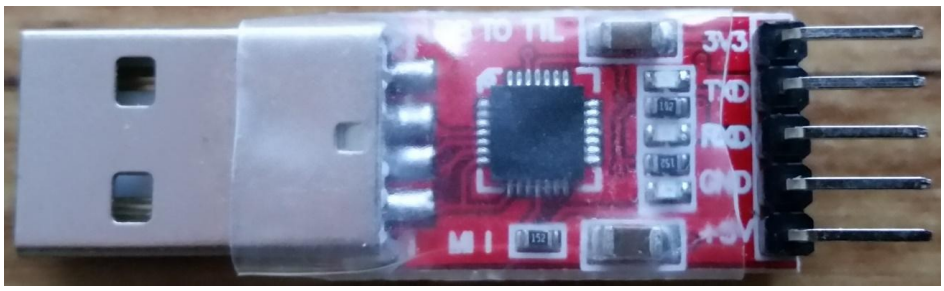


Рис. 1.48. Внутрішня структура та зовнішній вигляд модуля перетворювача UART-USB типу CP2102

PA2	USART2_TX/ TIM5_CH3/ TIM9_CH1/ TIM2_CH3/ ETH_MDIO/ ADC123_IN2
PA3	USART2_RX/ TIM5_CH4/ TIM9_CH2/ TIM2_CH4/ OTG_HS_ULPI_D0/ ETH_MII_COL/ ADC123_IN3

Рис. 1.49. Лінії TXD (порт PA2), RXD (порт PA3) USART2

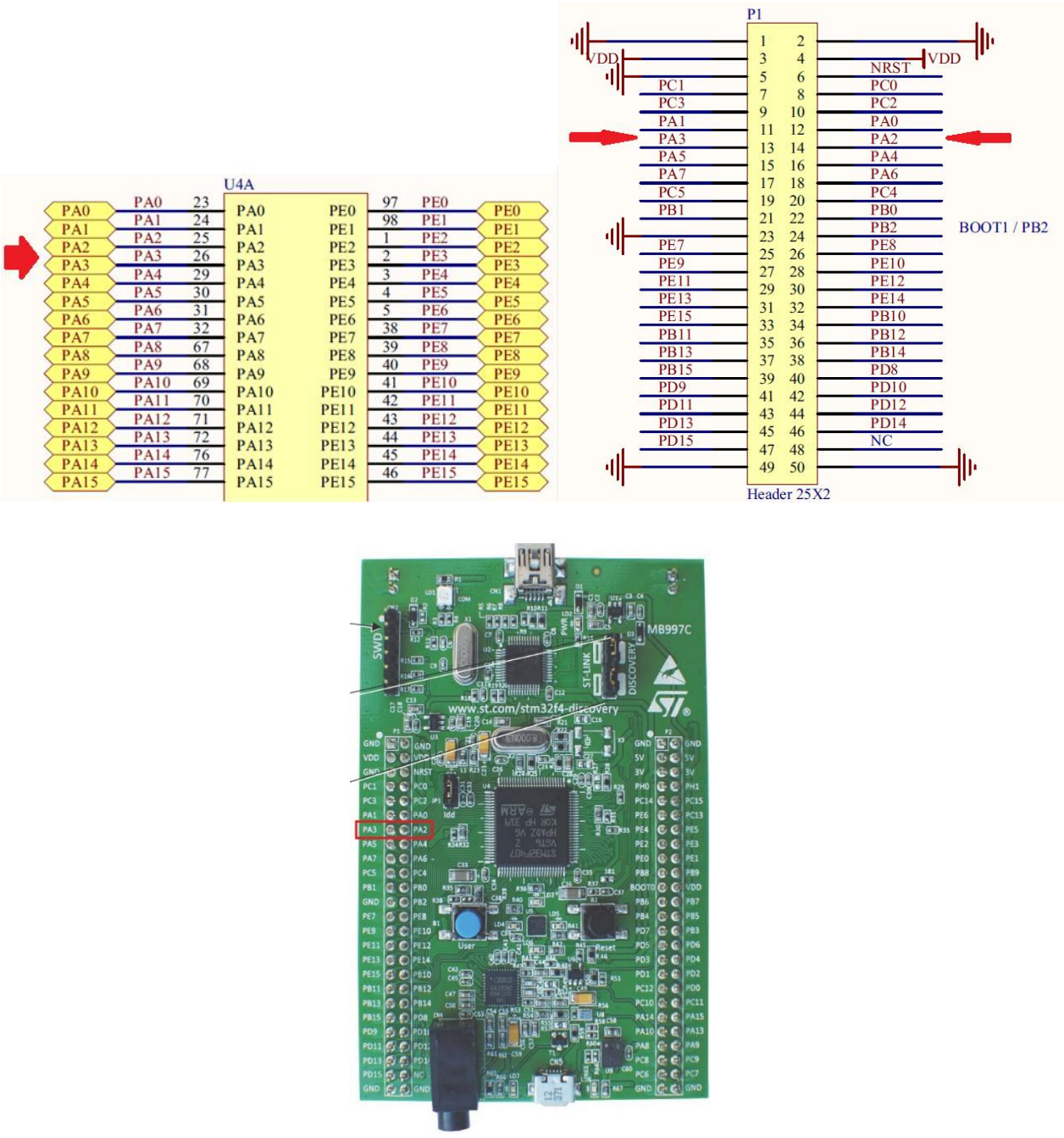


Рис. 1.50. Контакти підключення до ліній TXD (порт PA2), RXD (порт PA3) USART2 на платі STM32F4DISCOVERY

Література

1.1. RM0090 STM32F405xx_07xx Reference manual.pdf
1.2. STM32F407xx.pdf
1.3. STM32F4DISCOVERY User manual.pdf
1.4. hd44780u.pdf.
1.5. CP2102-9.pdf
1.6. Конспект лекцій з дисципліни “Мікропроцесорні системи”.

**Лістинг програмного модуля перетворення АЦП,
виводу результату на екран і передачі по UART**

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdio.h>
#include <stm32f4xx_hal_uart.h>

/* ADC handler declaration */
ADC_HandleTypeDef    AdcHandle;

/* Variable used to get converted value */
__IO uint16_t uhADCxConvertedValue = 0;

/* Private function prototypes -----------------------------------------------*/
static void SystemClock_Config(void);
static void Error_Handler(void);

/* Private functions ---------------------------------------------------------*/
//==============================================================================
//LCD1602
#define    LCM_OUT            GPIOB->ODR

#define    LCM_PIN_RS         GPIO_PIN_8       // PB8
#define    LCM_PIN_EN         GPIO_PIN_1       // PB1
#define    LCM_PIN_D7         GPIO_PIN_7       // PB7
#define    LCM_PIN_D6         GPIO_PIN_6       // PB6
#define    LCM_PIN_D5         GPIO_PIN_5       // PB5
#define    LCM_PIN_D4         GPIO_PIN_4       // PB4

#define    LCM_PIN_MASK  ((LCM_PIN_RS | LCM_PIN_EN | LCM_PIN_D7 | LCM_PIN_D6 |
LCM_PIN_D5 | LCM_PIN_D4))
GPIO_InitTypeDef  GPIO_InitStructure;

void delay(int a)
{
        int i = 0;
        int f = 0;
        while (f < a)
        {
                while (i < 60)
                    {i++; }
                f++;
        }
}

void PulseLCD()
```

```
{
	LCM_OUT &= ~LCM_PIN_EN;
	delay(10000);
	LCM_OUT |= LCM_PIN_EN;
	delay(10000);
	LCM_OUT &= (~LCM_PIN_EN);
	delay(10000);
}

//---Запис байта в дисплей---//
void SendByte(char ByteToSend, int IsData)
{
	LCM_OUT &= (~LCM_PIN_MASK);
	LCM_OUT |= (ByteToSend & 0xF0);

	if (IsData == 1)
		LCM_OUT |= LCM_PIN_RS;
	else
		LCM_OUT &= ~LCM_PIN_RS;
	PulseLCD();
	LCM_OUT &= (~LCM_PIN_MASK);
	LCM_OUT |= ((ByteToSend & 0x0F) << 4);

	if (IsData == 1)
		LCM_OUT |= LCM_PIN_RS;
	else
		LCM_OUT &= ~LCM_PIN_RS;

	PulseLCD();
}

//---Установка позиції курсора---//
void Cursor(char Row, char Col)
{
	char address;
	if (Row == 0)
		address = 0;
	else
		address = 0x40;
	address |= Col;
	SendByte(0x80 | address, 0);
}

//---Очистка дисплея---//
void ClearLCDScreen()
{
	SendByte(0x01, 0);
	SendByte(0x02, 0);
}
```

```
//---Ініціалізація дисплея---//

static void InitLCD(void);
static void InitLCD(void)
{
	__GPIOB_CLK_ENABLE();

	GPIO_InitTypeDef GPIO_InitStructure;


	GPIO_InitStructure.Pin   = GPIO_PIN_1 | GPIO_PIN_4 | GPIO_PIN_5 |
				   GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8;
	GPIO_InitStructure.Mode  = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
	GPIO_InitStructure.Pull  = GPIO_NOPULL;
	HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);

	LCM_OUT &= ~(LCM_PIN_MASK);
	delay(10000);

	LCM_OUT &= ~LCM_PIN_RS;
	LCM_OUT &= ~LCM_PIN_EN;
	LCM_OUT = 0x20;
	PulseLCD();
	SendByte(0x28, 0);
	SendByte(0x0E, 0);
	SendByte(0x06, 0);
}
//---Вивід рядка---//
void PrintStr(char *Text)
{
	char *c;
	c = Text;
	while ((c != 0) && (*c != 0))
	{
		SendByte(*c, 1);
		c++;
	}
}

char str[17];
float t;
//=====================================================
//UART
uint8_t aTxBuffer[] = { 0x01 };
UART_HandleTypeDef UartHandle;

/* Size of Transmission buffer */
//#define TXBUFFERSIZE                      (COUNTOF(aTxBuffer) - 1)
```

```
#define TXBUFFERSIZE                    (COUNTOF(aTxBuffer))
/* Size of Reception buffer */
#define RXBUFFERSIZE                    TXBUFFERSIZE

/* Exported macro ------------------------------------------------------------*/
#define COUNTOF(__BUFFER__)   (sizeof(__BUFFER__) / sizeof(*(__BUFFER__)))

/* User can use this section to tailor USARTx/UARTx instance used and associated
   resources */
/* Definition for USARTx clock resources */
#define USARTx                          USART2
#define USARTx_CLK_ENABLE()             __HAL_RCC_USART2_CLK_ENABLE();
#define USARTx_RX_GPIO_CLK_ENABLE()     __HAL_RCC_GPIOA_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE()     __HAL_RCC_GPIOA_CLK_ENABLE()

#define USARTx_FORCE_RESET()            __HAL_RCC_USART2_FORCE_RESET()
#define USARTx_RELEASE_RESET()          __HAL_RCC_USART2_RELEASE_RESET()

/* Definition for USARTx Pins */
#define USARTx_TX_PIN                   GPIO_PIN_2
#define USARTx_TX_GPIO_PORT             GPIOA
#define USARTx_TX_AF                    GPIO_AF7_USART2
#define USARTx_RX_PIN                   GPIO_PIN_3
#define USARTx_RX_GPIO_PORT             GPIOA
#define USARTx_RX_AF                    GPIO_AF7_USART2

/*##-1- Configure the UART peripheral ######################################*/
 /* Put the USART peripheral in the Asynchronous mode (UART Mode) */
 /* UART1 configured as follow:
    - Word Length = 8 Bits
    - Stop Bit = One Stop bit
    - Parity = None
    - BaudRate = 9600 baud
    - Hardware flow control disabled (RTS and CTS signals) */

void InitUART(void)
{
        UartHandle.Instance         = USARTx;

        UartHandle.Init.BaudRate    = 9600;
        UartHandle.Init.WordLength  = UART_WORDLENGTH_8B;
        UartHandle.Init.StopBits    = UART_STOPBITS_1;
        UartHandle.Init.Parity      = UART_PARITY_NONE;
        UartHandle.Init.HwFlowCtl   = UART_HWCONTROL_NONE;
        UartHandle.Init.Mode        = UART_MODE_TX_RX;
        UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

        if (HAL_UART_Init(&UartHandle) != HAL_OK)
        {
```

```
            Error_Handler();
        }


}

//=================================================
int main(void)
{
        ADC_ChannelConfTypeDef sConfig;

        /* STM32F4xx HAL library initialization:
            - Configure the Flash prefetch, instruction and Data caches
            - Configure the Systick to generate an interrupt each 1 msec
            - Set NVIC Group Priority to 4
            - Global MSP (MCU Support Package) initialization
        */
        HAL_Init();

        /* Configure the system clock to 144 MHz */
        SystemClock_Config();

        InitLCD();                  //Ініціалізація дисплея
delay(500);

        InitUART();  //Ініціалізація UART

  /* Configure LED4 and LED5 */
        //BSP_LED_Init(LED4);
        //BSP_LED_Init(LED5);

        /*##-1- Configure the ADC peripheral #######################################*/
        AdcHandle.Instance = ADCx;

        AdcHandle.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
        AdcHandle.Init.Resolution = ADC_RESOLUTION_12B;
        AdcHandle.Init.ScanConvMode = DISABLE;
        AdcHandle.Init.ContinuousConvMode = ENABLE;
        AdcHandle.Init.DiscontinuousConvMode = DISABLE;
        AdcHandle.Init.NbrOfDiscConversion = 0;
        AdcHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
        AdcHandle.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_CC1;
        AdcHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;
        AdcHandle.Init.NbrOfConversion = 1;
        //  AdcHandle.Init.DMAContinuousRequests = ENABLE;
          AdcHandle.Init.DMAContinuousRequests = DISABLE;
        AdcHandle.Init.EOCSelection = DISABLE;

        if (HAL_ADC_Init(&AdcHandle) != HAL_OK)
```

```
        {
                /* Initialization Error */
                Error_Handler();
        }

        /*##-2- Configure ADC regular channel ######################################*/
        /* Note: Considering IT occurring after each number of size of            */
        /*       "uhADCxConvertedValue"  ADC conversions (IT by DMA end           */
        /*       of transfer), select sampling time and ADC clock with sufficient */
        /*       duration to not create an overhead situation in IRQHandler.      */
        sConfig.Channel = ADCx_CHANNEL;
        sConfig.Rank = 1;
        sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
        sConfig.Offset = 0;

        if (HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
        {
                /* Channel Configuration Error */
                Error_Handler();
        }


        ClearLCDScreen();            //Очистка дисплея
delay(500);
        Cursor(0, 2);               //Установка курсора в рядку 1
delay(500);
        PrintStr("  Test LCD");       //Вивід тексту
delay(500);
        Cursor(1, 4);                 //Установка курсора в рядку 2
delay(500);
        PrintStr("BC1602AG");
        SendByte(0b00001100, 0);    //Курсор виключено

  /* Infinite loop */
  while(1)
        {
                a = 0;

                if (HAL_ADC_Start_DMA(&AdcHandle, (uint32_t*)&uhADCxConvertedValue, 1) !=
HAL_OK)
                {
                        /* Start Conversation Error */
                        Error_Handler();
                }

                Cursor(0, 1);                   //Установка курсора в рядку 1
                delay(500);
                PrintStr("0.1 Meas Ua, B");     //Вивід тексту
                delay(500);
```

```
                Cursor(1, 4);                   //Установка курсора в рядку 2
                delay(500);

                snprintf(str, 17, " %d          ", uhADCxConvertedValue);
                PrintStr(str);

                if (HAL_ADC_Stop_DMA(&AdcHandle) != HAL_OK)
                {
                        /* Start Conversation Error */
                        Error_Handler();
                }

                aTxBuffer[0] = (uint8_t)(uhADCxConvertedValue & 0x00ff);

                if (HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)
!= HAL_OK)
                {
                        Error_Handler();
                }

                aTxBuffer[0] = (uint8_t)(uhADCxConvertedValue & 0xff00);

                if (HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)
!= HAL_OK)
                {
                        Error_Handler();
                }

        }
}


/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follow :
  *           System Clock source           = PLL (HSE)
  *           SYSCLK(Hz)                    = 144000000
  *           HCLK(Hz)                      = 144000000
  *           AHB Prescaler                 = 1
  *           APB1 Prescaler                = 4
  *           APB2 Prescaler                = 2
  *           HSE Frequency(Hz)             = 8000000
  *           PLL_M                         = 8
  *           PLL_N                         = 288
  *           PLL_P                         = 2
  *           PLL_Q                         = 6
  *           VDD(V)                        = 3.3
  *           Main regulator output voltage  = Scale2 mode
  *           Flash Latency(WS)             = 4
```

```
  * @param  None
  * @retval None
  */
static void SystemClock_Config(void)
{
        RCC_ClkInitTypeDef RCC_ClkInitStruct;
        RCC_OscInitTypeDef RCC_OscInitStruct;

        /* Enable Power Control clock */
        __HAL_RCC_PWR_CLK_ENABLE();

        /* The voltage scaling allows optimizing the power consumption when the device is
          clocked below the maximum system frequency, to update the voltage scaling value
          regarding system frequency refer to product datasheet.  */
        __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

        /* Enable HSE Oscillator and activate PLL with HSE as source */
        RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
        RCC_OscInitStruct.HSEState = RCC_HSE_ON;
        RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
        RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
        RCC_OscInitStruct.PLL.PLLM = 8;
        RCC_OscInitStruct.PLL.PLLN = 288;
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
        RCC_OscInitStruct.PLL.PLLQ = 6;
        HAL_RCC_OscConfig(&RCC_OscInitStruct);

        /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
          clocks dividers */
        RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
        RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
        RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
        RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
        RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
```

Лабораторна робота № 2

ДОСЛІДЖЕННЯ ВУЗЛА ВИВЕДЕННЯ ГРАФІЧНОЇ ІНФОРМАЦІЇ В МПС

МЕТА РОБОТИ: Дослідження вузла виведення графічної інформації на графічний індикатор. Підготовка в середовищі STM32CubeMX базової програми і драйверів вузлів виводу інформації на одиночні світлодіодні індикатори та графічної інформації на кольоровий графічний індикатор з використанням відлагоджувального модуля STM32F4DISCOVERY.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з лабораторним стендом ***на відлагоджувального модуля stm32F407***
2. Виконати у вказаному порядку основні етапи підготовки програми керування світлодіодами з використанням STM32CubeMX (**ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ**).
3. Запрограмувати з допомогою утиліти **STM32 ST-LINK Utility** тестову програму **tft_lcd_spi.hex** для виводу символьної та графічної інформації на кольоровий графічний індикатор **ILI9341**.
4. Використовуючи програмні модулі керування графічним індикатором **ILI9341** згідно до завдання керівника підготувати програму виводу символьної та графічної інформації на графічний індикатор.
5. Відповісти на контрольні запитання.
6. Захистити лабораторну роботу.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Ознайомитися з методичними матеріалами до лабораторної роботи
2. Скачати STM32CubeMX з сайту https://my.st.com/cas/login?service=https%3A%2F%2Fmy.st.com%2Fcontent%2Fmy_st_com%2Fen%2Fproducts%2Fdevelopment-tools%2Fsoftware-development-tools%2Fstm32-software-development-tools%2Fstm32-configurators-and-code-generators%2Fstm32cubemx.html

Для цього необхідно відкрити на вказаному сайті власний Account (**Create Account**)

3. Заінсталювати STM32CubeMX на власному ПК.
4. Ознайомитися з пакетом STM32CubeMX

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Основні компоненти лабораторного стенда.
2. Схема підключення інтегрованих на платі стенда світлодіодів.
3. Основні параметри графічного індикатора **ILI9341.**
4. Схема підключення графічного індикатора **ILI9341** до відлагоджувального модуля **STM32F4DISCOVERY** лабораторного стенда**.**
5. Основні етапи підготовки програми в STM32CubeMX
6. Назвати засоби програмування пам’яті на кристалі мікропроцесорного компонента
7. Пояснити роботу підготовленої та відлагодженої програми.

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

1. Запустити **STM32CubeMX:**

Рис. 2.1. Вікно запуску STM32CubeMX

2. **Поетапно** виконати кроки по підготовці для IDE Keil 4 програмного коду “мигання” світлодіодами на відлагоджувальному модулі STM32F4DISCOVERY:

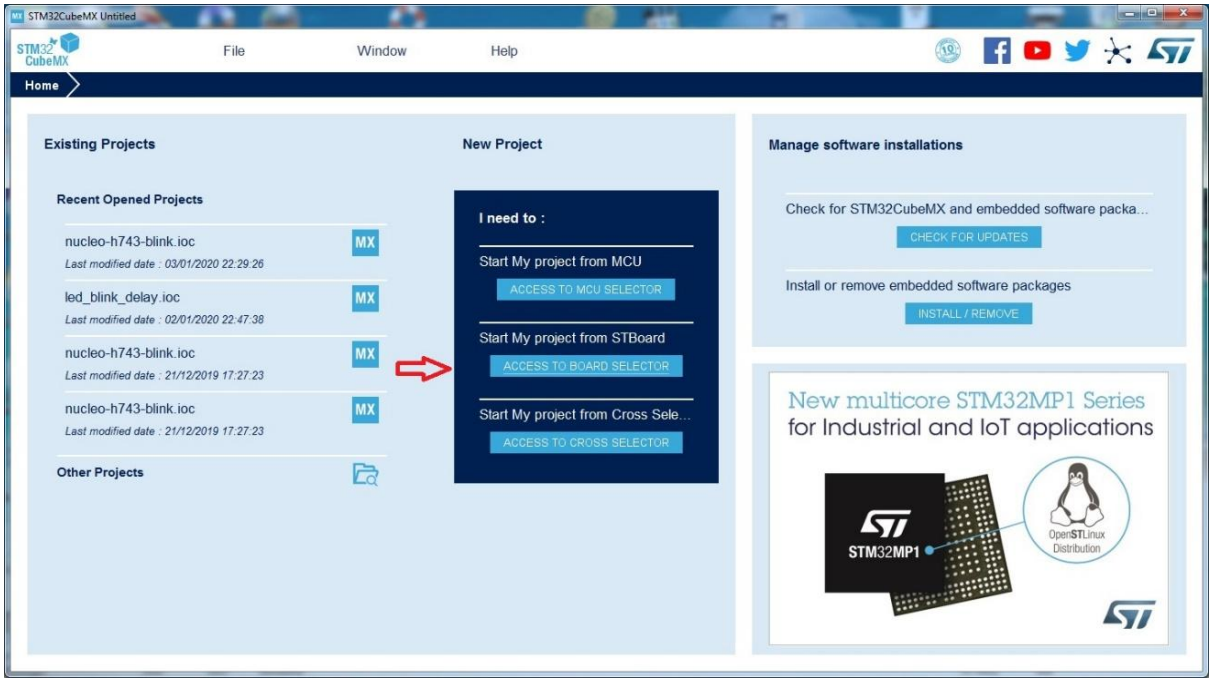


Рис. 2.2. Вибір апаратного відлагоджувального модуля

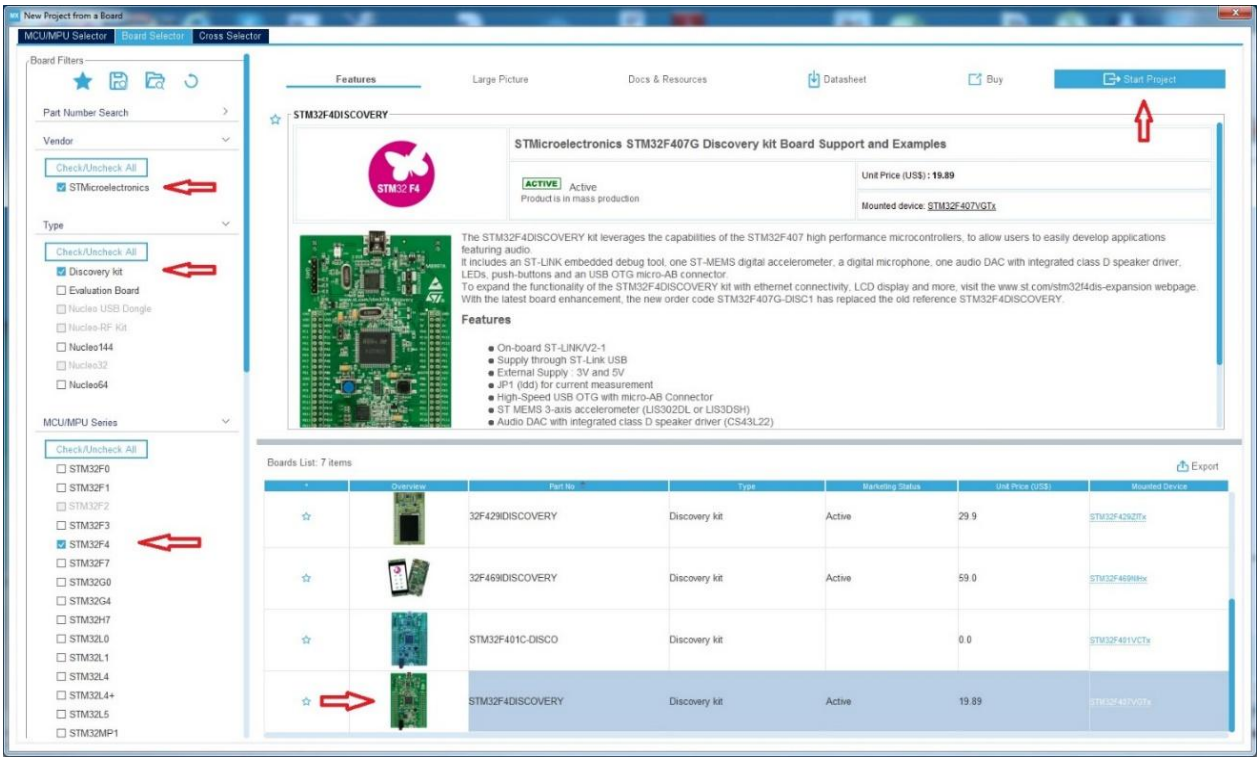


Рис. 2.3. Вибір апаратного відлагоджувального модуля STM32F4DISCOVERY

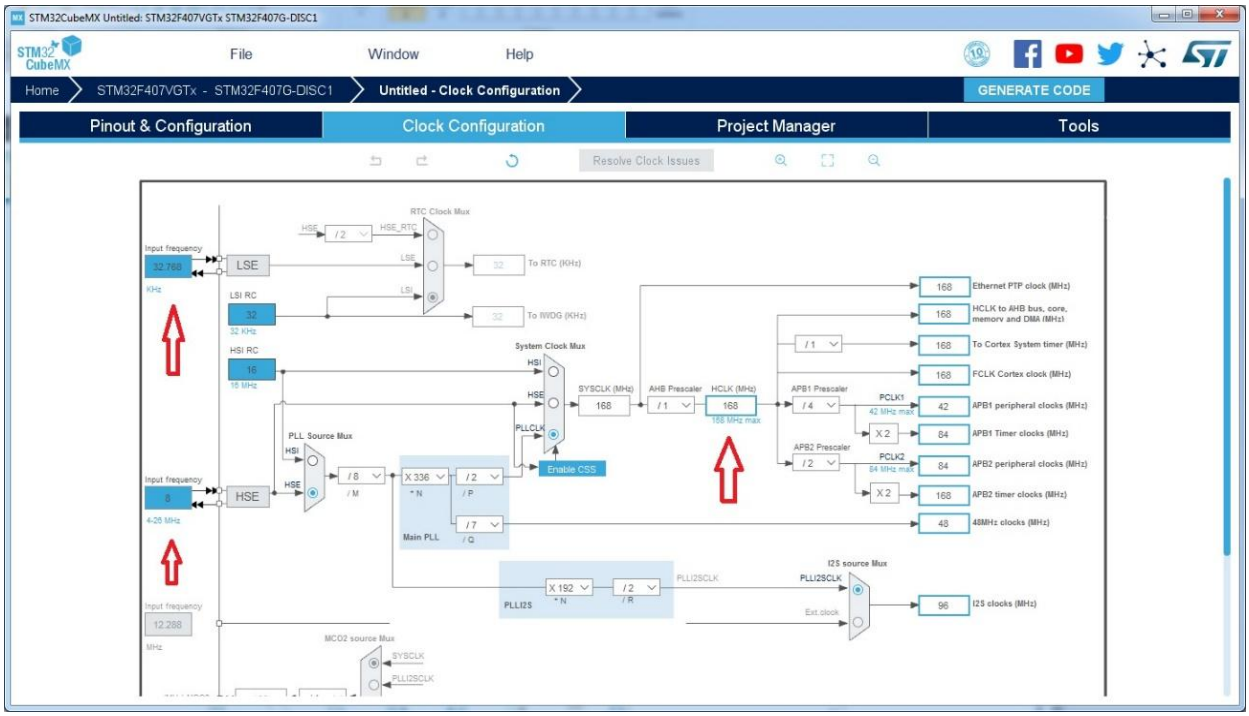


Рис. 2.4. Вибір параметрів системи синхронізації

Вибір та перевірка ліній I/O керування світлодіодами та при необхідності ліній інших вузлів мікропроцесорної системи відповідного функціонального призначення

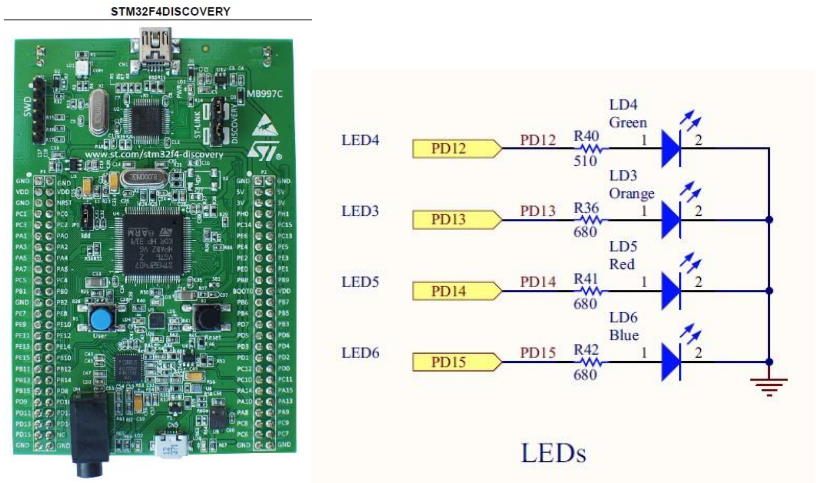


Рис. 2.5. Світлодіоди на відлагоджувальній платі

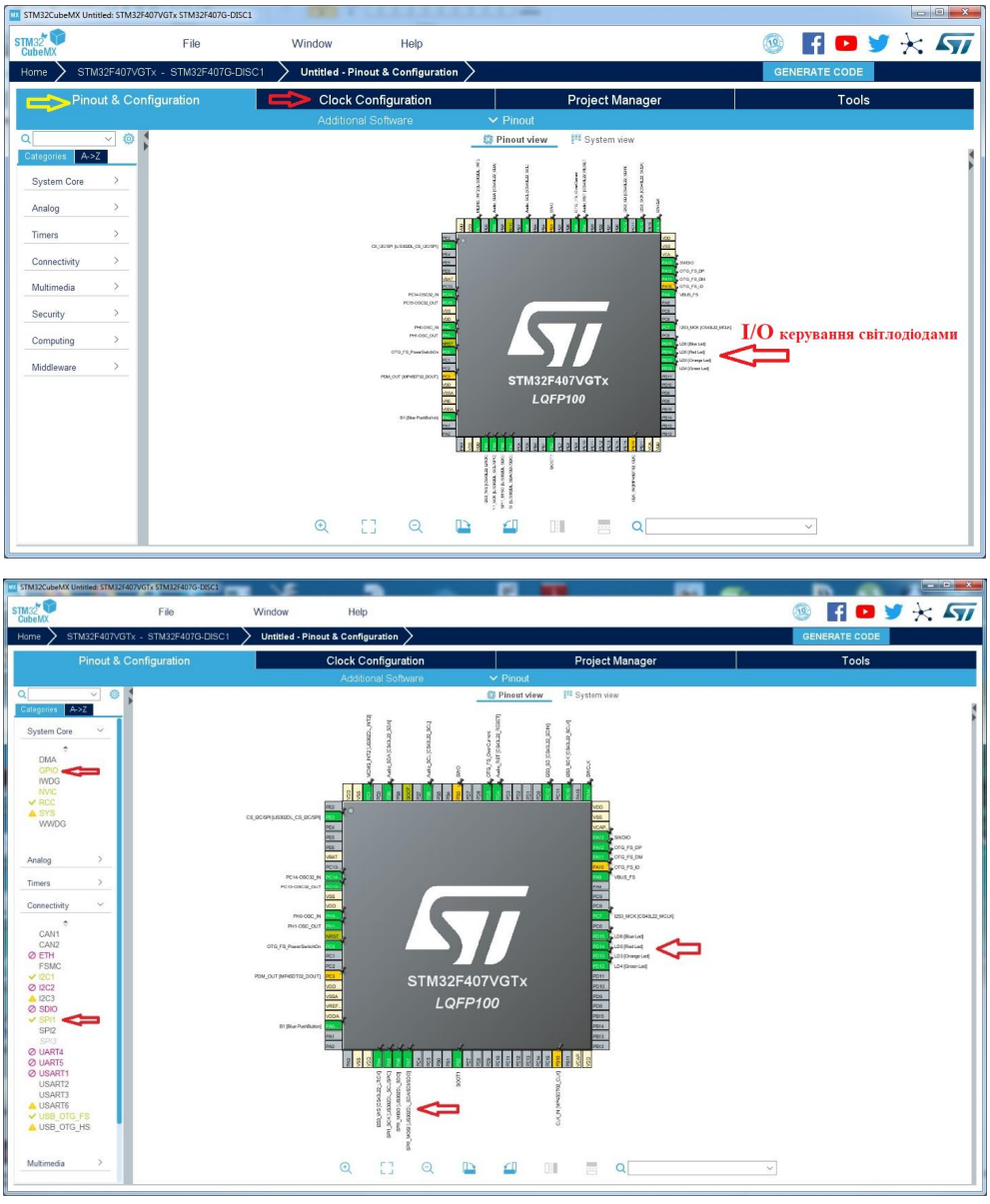


Рис. 2.6. Вибір та перевірка ліній I/O керування світлодіодами

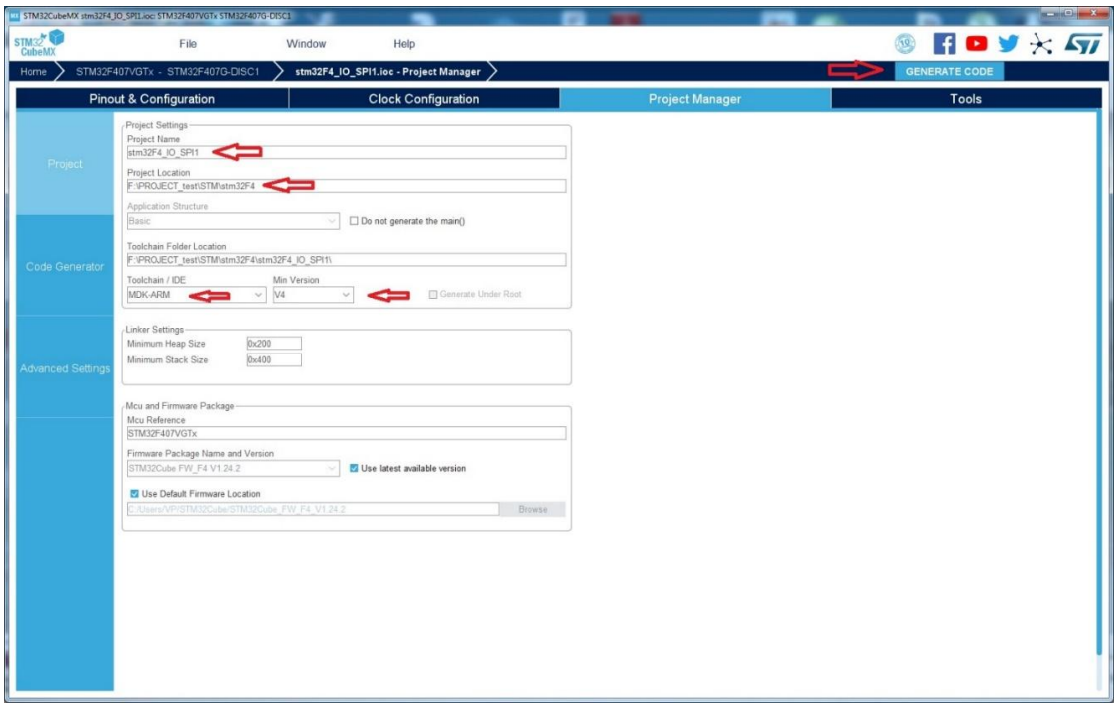


Рис. 2.7. Вибір назви проєкту та середовища IDE для генерування коду

Рис. 2.8. процес генерування коду

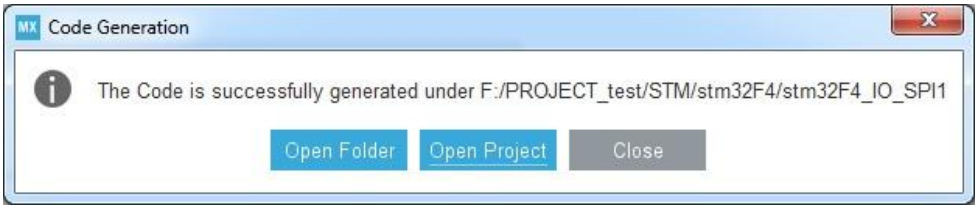


Рис. 2.9. запуск проєкту

STM32CubeMX

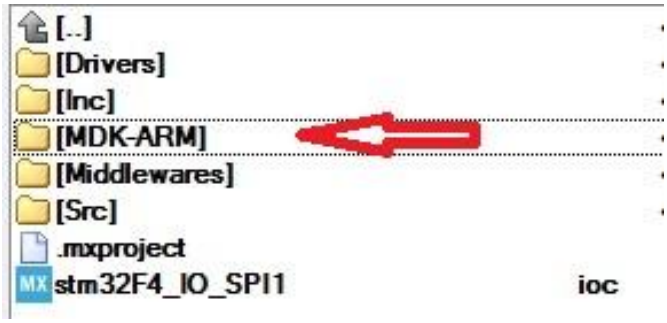


Рис. 2.10. розміщення згенерованого IDE проєкту в папці проєкту

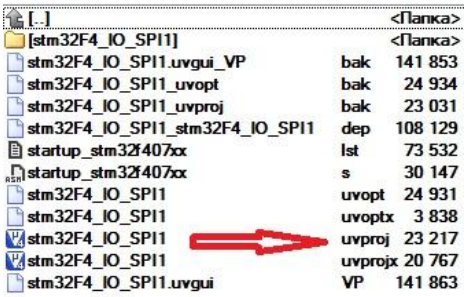


Рис. 2.11. файл запуску IDE проєкту

Процес запуску IDE проєкту

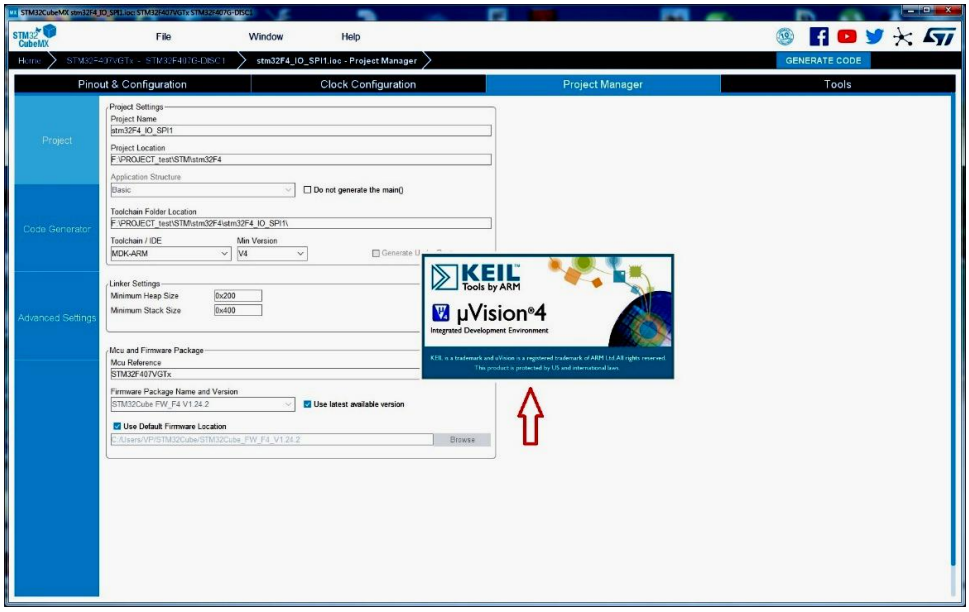


Рис. 2.12. Процес запуску IDE проєкту

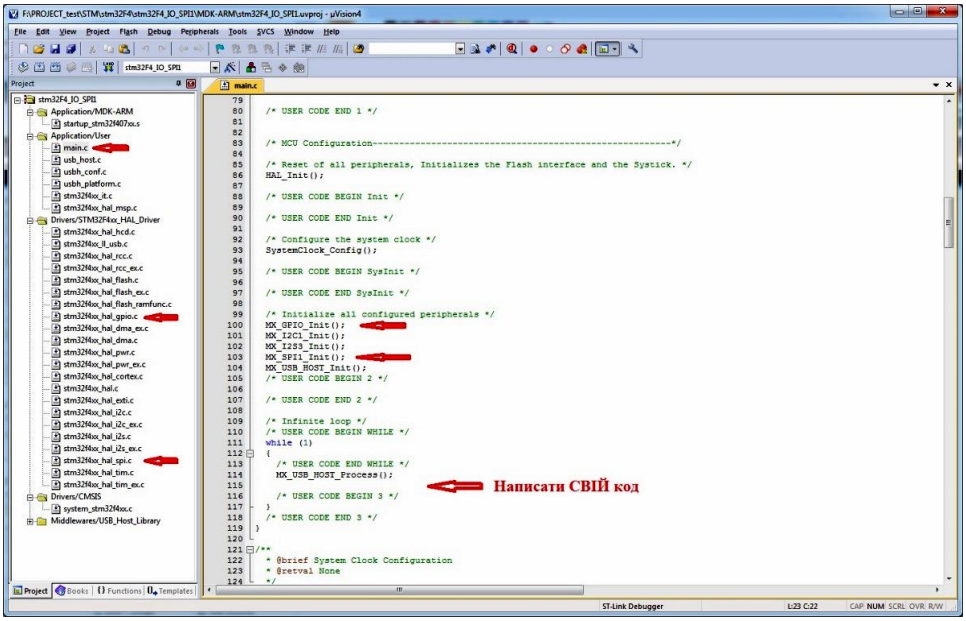


*Рис. 2.13. Вікно модуля **main.c***

Запуск компіляції, виправлення помилок при їх наявності, перевірка та при необхідності корекція відповідних модулів ініціалізації I/O та інших вузлів у відповідності до мікропроцесорної системи конкретного функціонального призначення

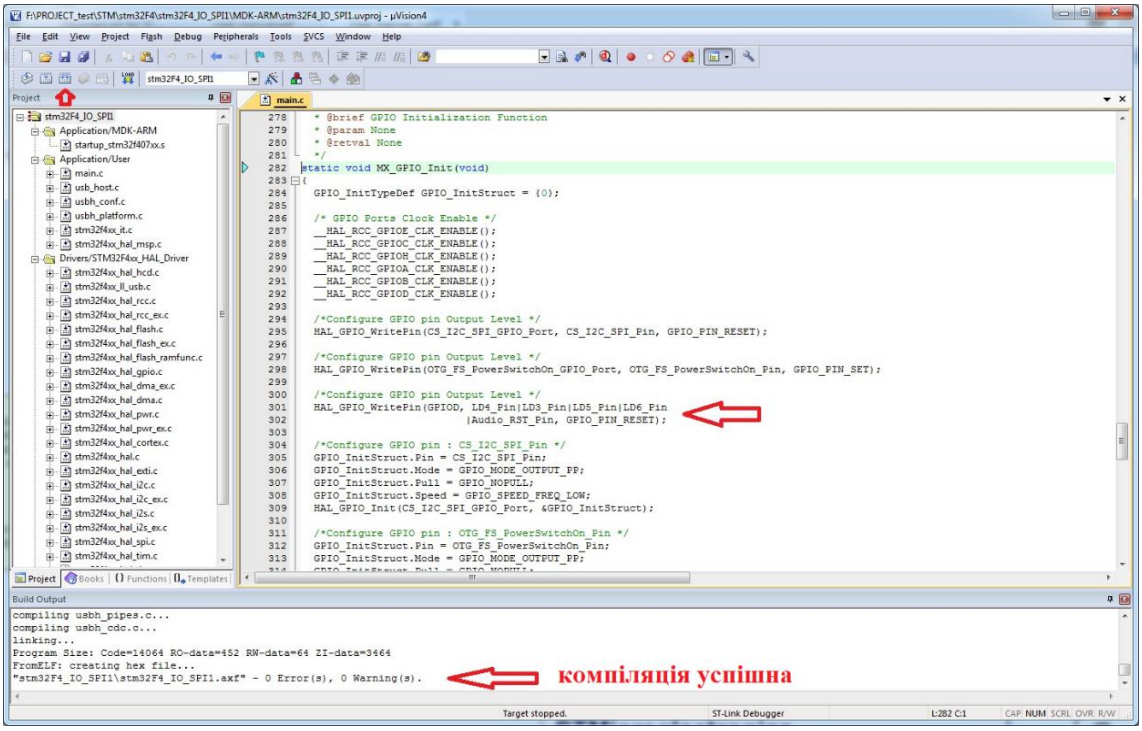


Рис. 2.14. Запуск компіляції

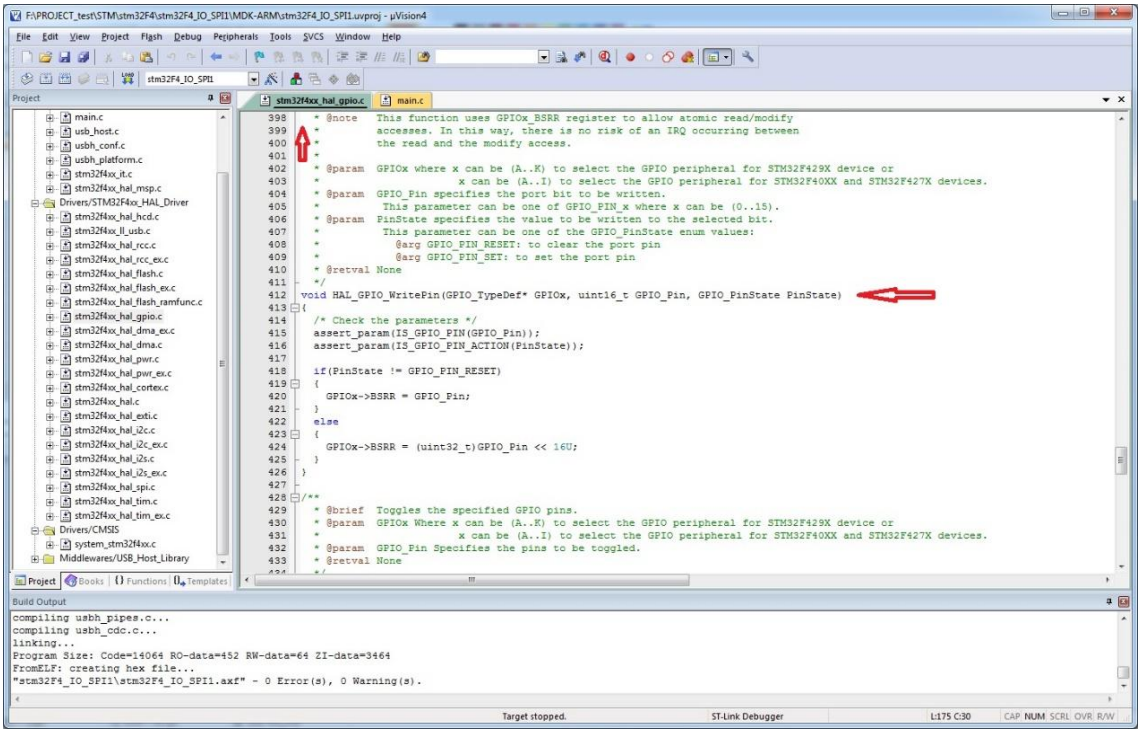


Рис. 2.15. Модуль керування лініями I/O

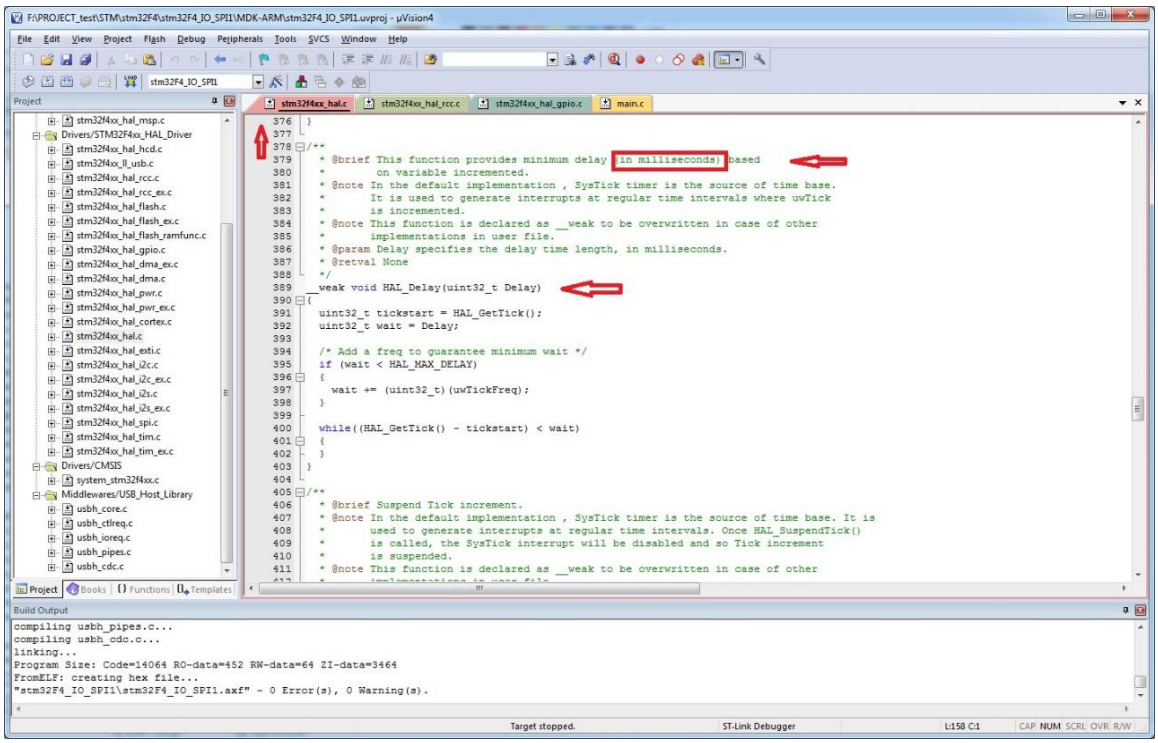


Рис. 2.16. Модуль програмної затримки

Підключити стенд до гнізда USB на ПК та запустити відлагоджувач

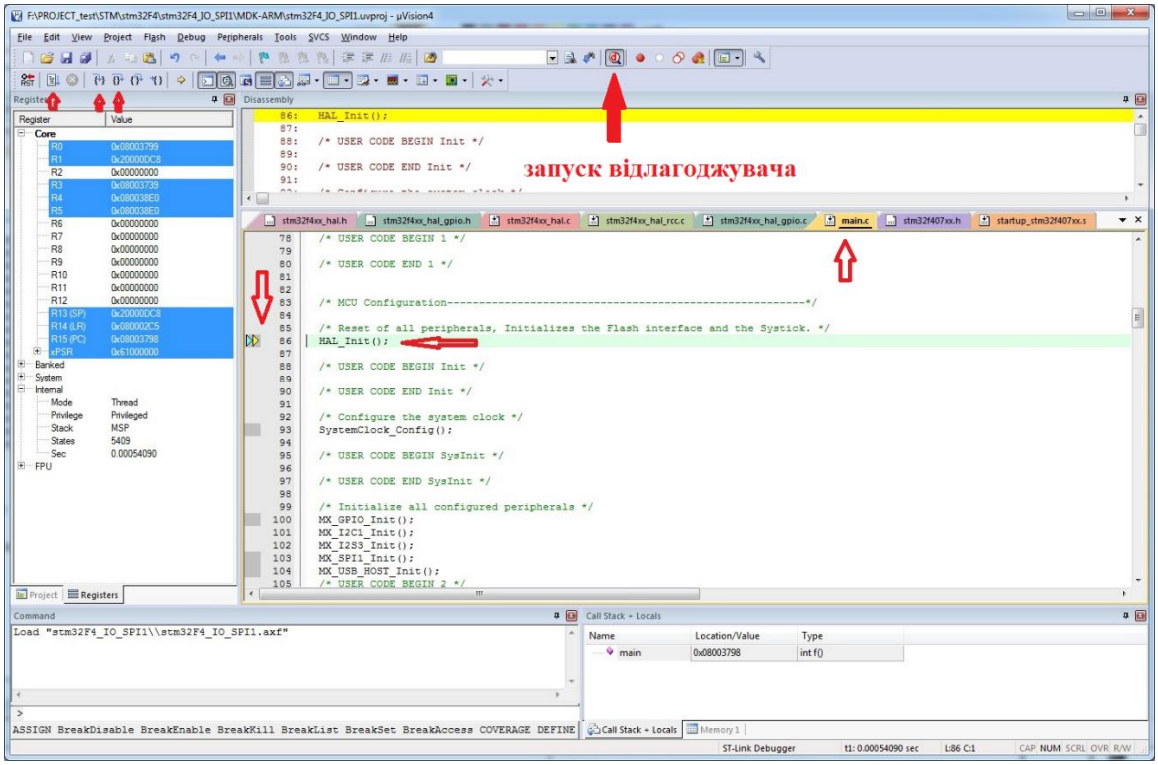


*Рис. 2.17. Вікно **main.c** після вдалого запуску*

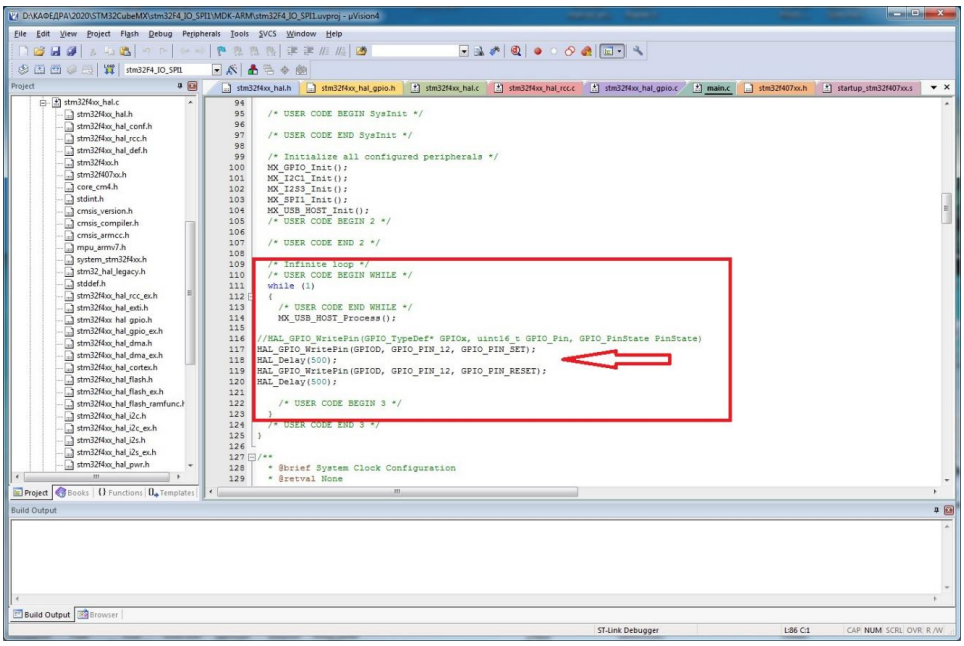


Рис. 2.18. Керування світлодіодом з використанням затримки

3. Запис завантажувального модуля програми в форматі *.hex в пам'ять на кристалі мікропроцесорного компонента

Програмування пам'яті на кристалі мікропроцесорного компонента можна здійснювати різними засобами:

– в процесі запуску відлагоджувача в режимі роботи з апаратурою;

– інтегрованим програматором в середовище IDE;

– окремою програмою, призначеною для роботи з відповідним мікропроцесорним сімейством, наприклад, утилітою **STM32 ST-LINK Utility.exe**:

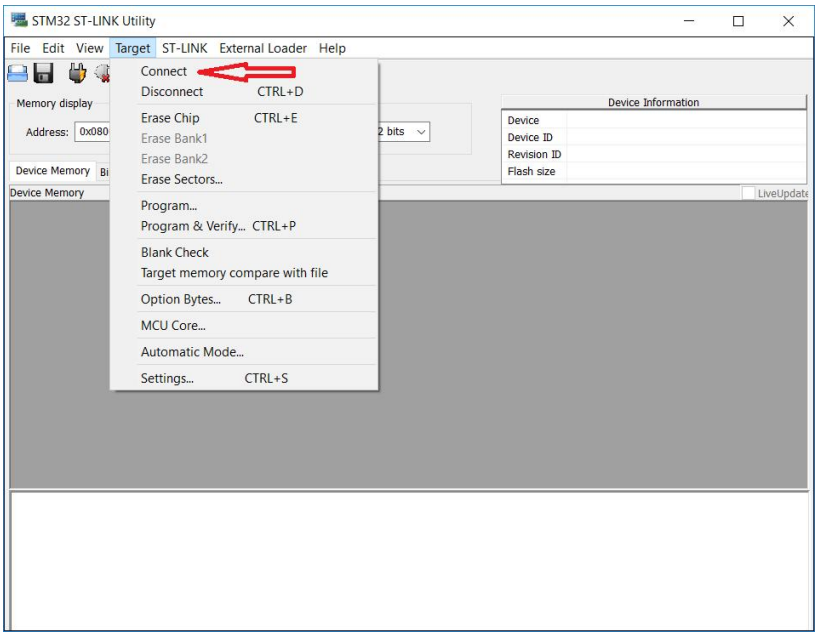


*Рис. 2.19. Вікно запуску **STM32 ST-LINK Utility***

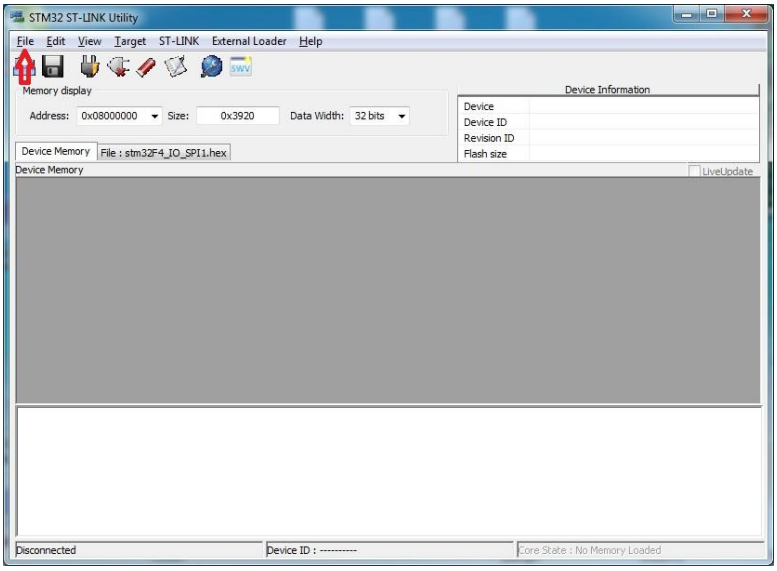


Рис. 2.19 (продовження). Вікно запуску ***STM32 ST-LINK Utility***

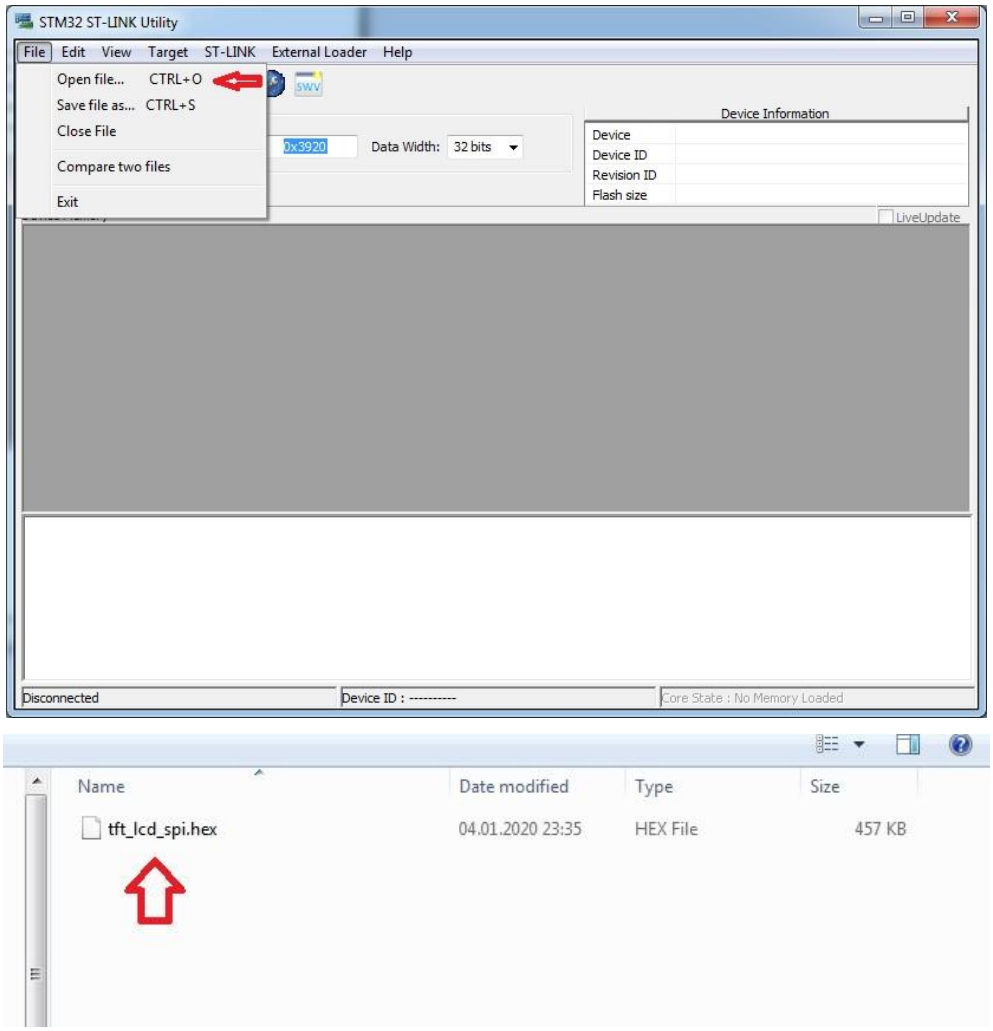


Рис. 2.20. Вибір файла ***в форматі *.hex*** *для програмування в пам'ять кристалу*

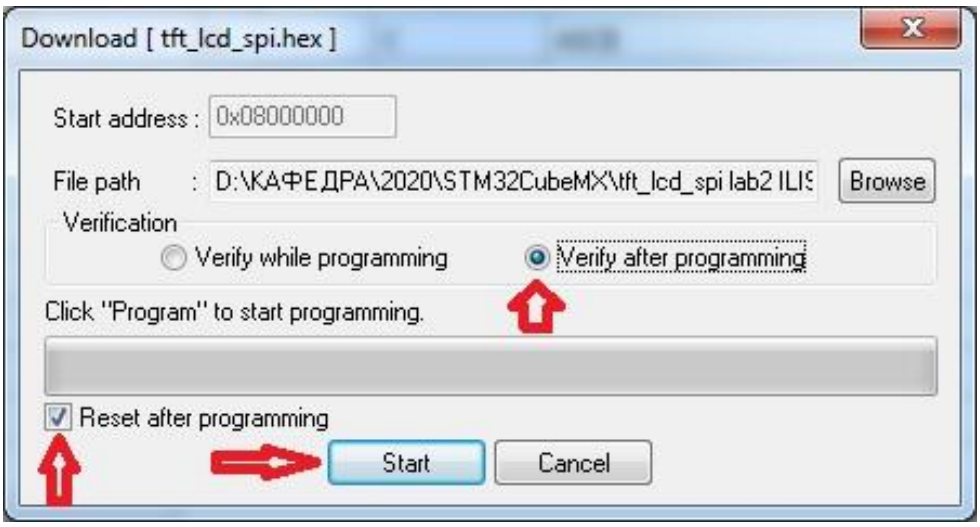


Рис. 2.21. Вибір режимів програмування
*та запуск процесу програмування "**Start**"*

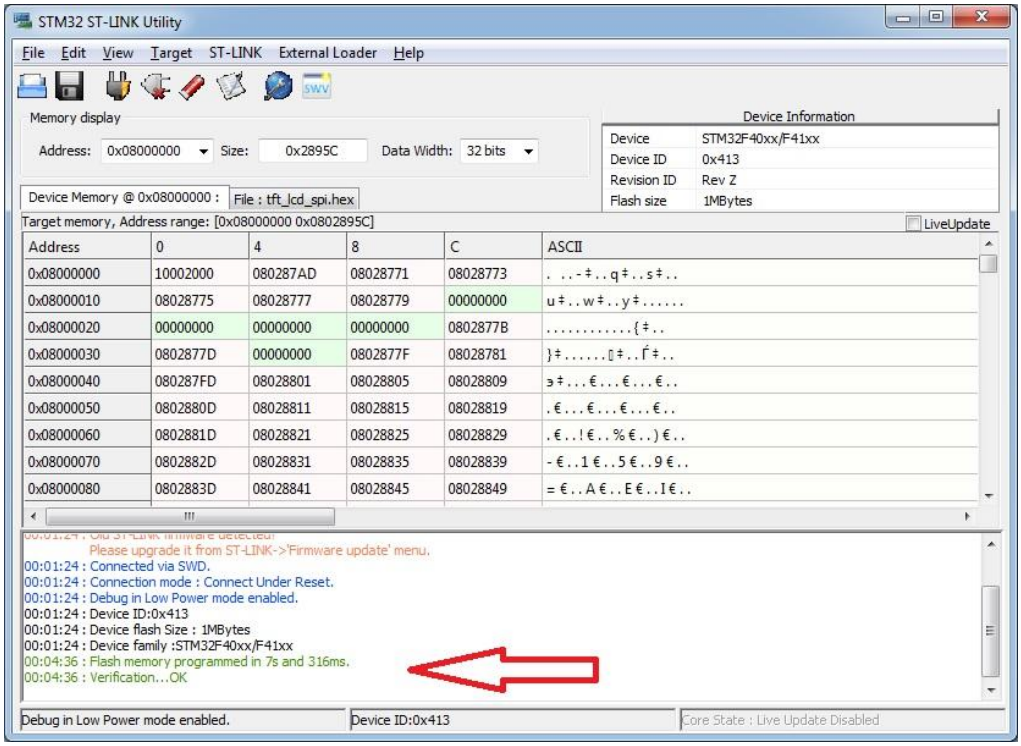


Рис. 2.22. Успішне завершення процесу програмування
*та верифікації запрограмованого коду з вхідним ***.hex***

МЕТОДИЧНІ МАТЕРІАЛИ

Лабораторний стенд

Стенд реалізовано на базі відлагоджувального модуля **STM32F4DISCOVERY** з підключеним через порт **SPI** кольоровим графічним індикатором типу **ILI9341**. Підключення стенду здійснюється **USB** кабелем до любого порту **USB 2.0** чи **USB 3.0** комп'ютера. Світлодіоди кольорові і знаходяться на платі **STM32F4DISCOVERY.**

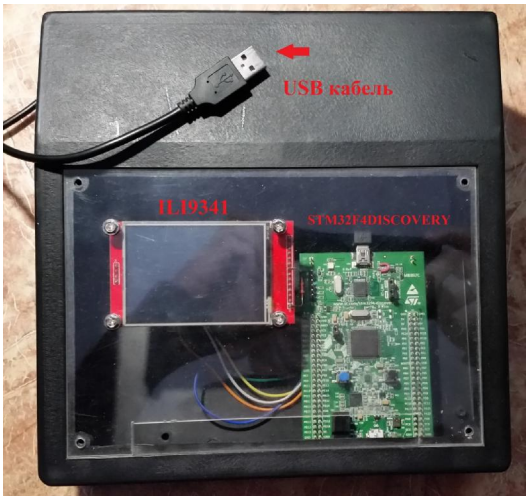


Рис. 2.23. Лабораторний стенд

3. Відлагоджувальний модуль STM32F4DISCOVERY

Відлагоджувальний модуль **STM32F4DISCOVERY** призначений для відлагодження в реальному часі програм для мікропроцесорних систем (МПС) різного функціонального призначення з метою розпаралелювання процесу розроблення апаратної частини МПС та програмного забезпечення до неї. **STM32F4DISCOVERY** реалізовано на базі мікроконтролера з ядром **Cortex-M4F** типу **STM32F407VGT6** в корпусі **LQFP100**. Структурна схема та зовнішній вигляд **STM32F4DISCOVERY** зображені на рис.

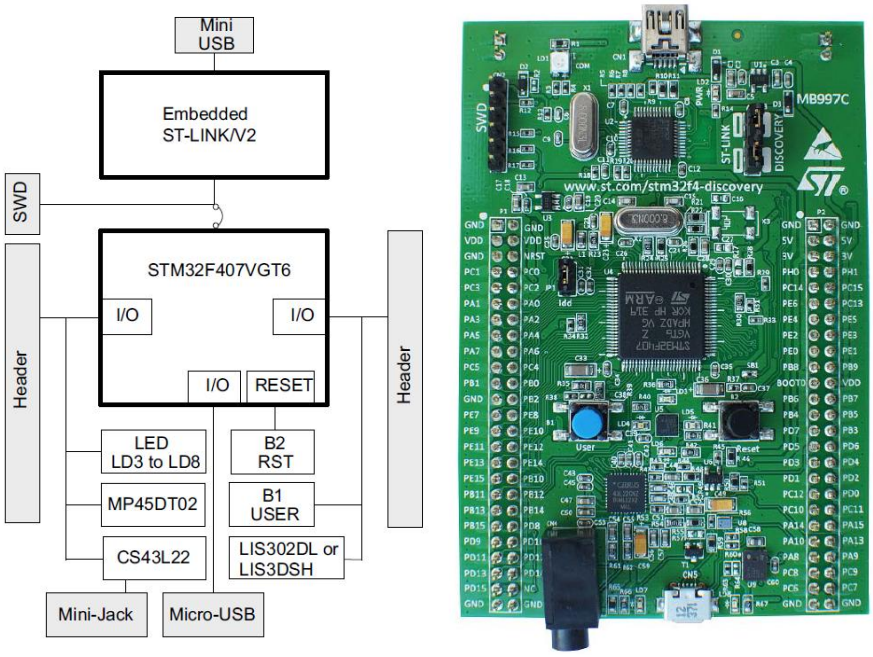
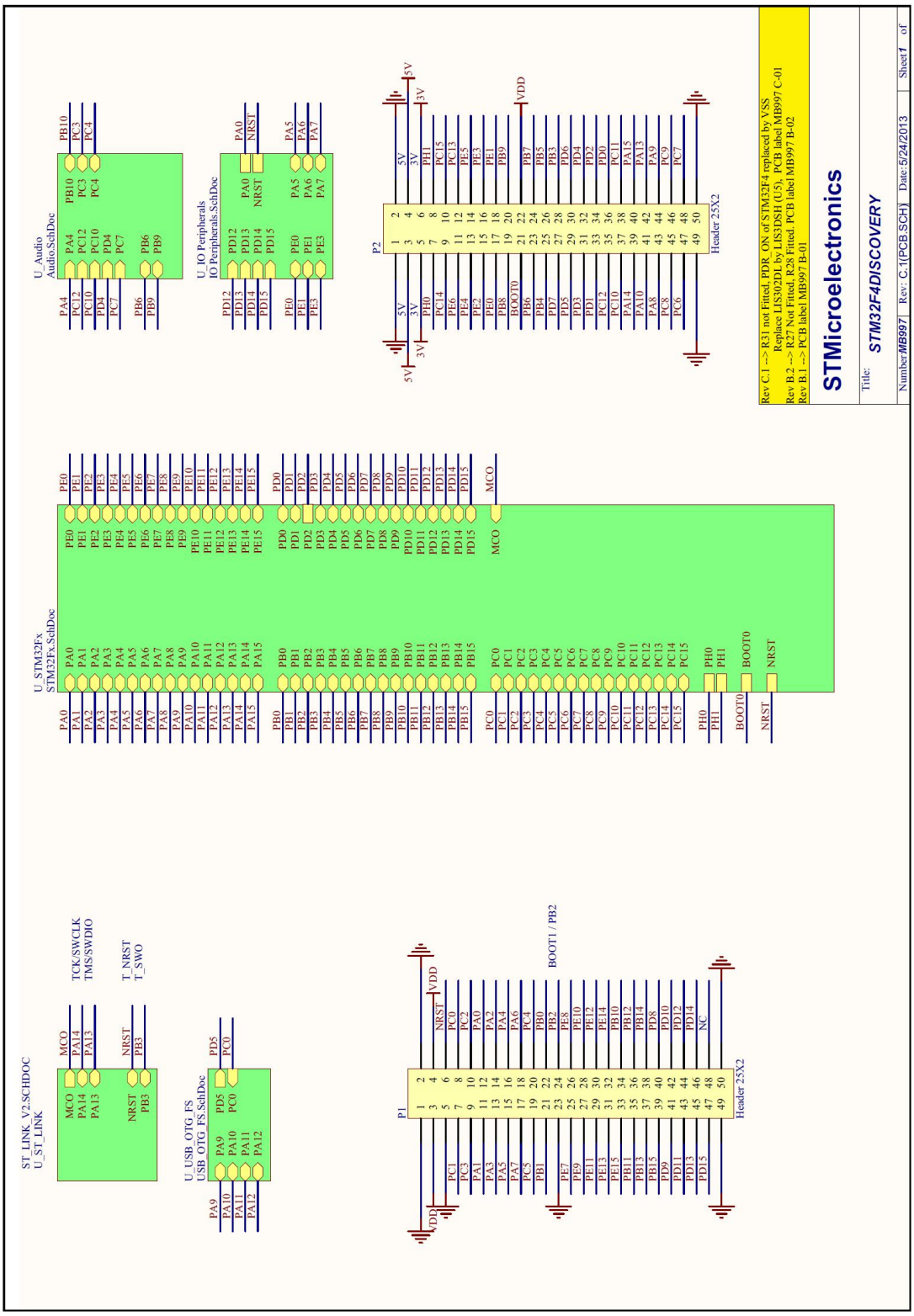


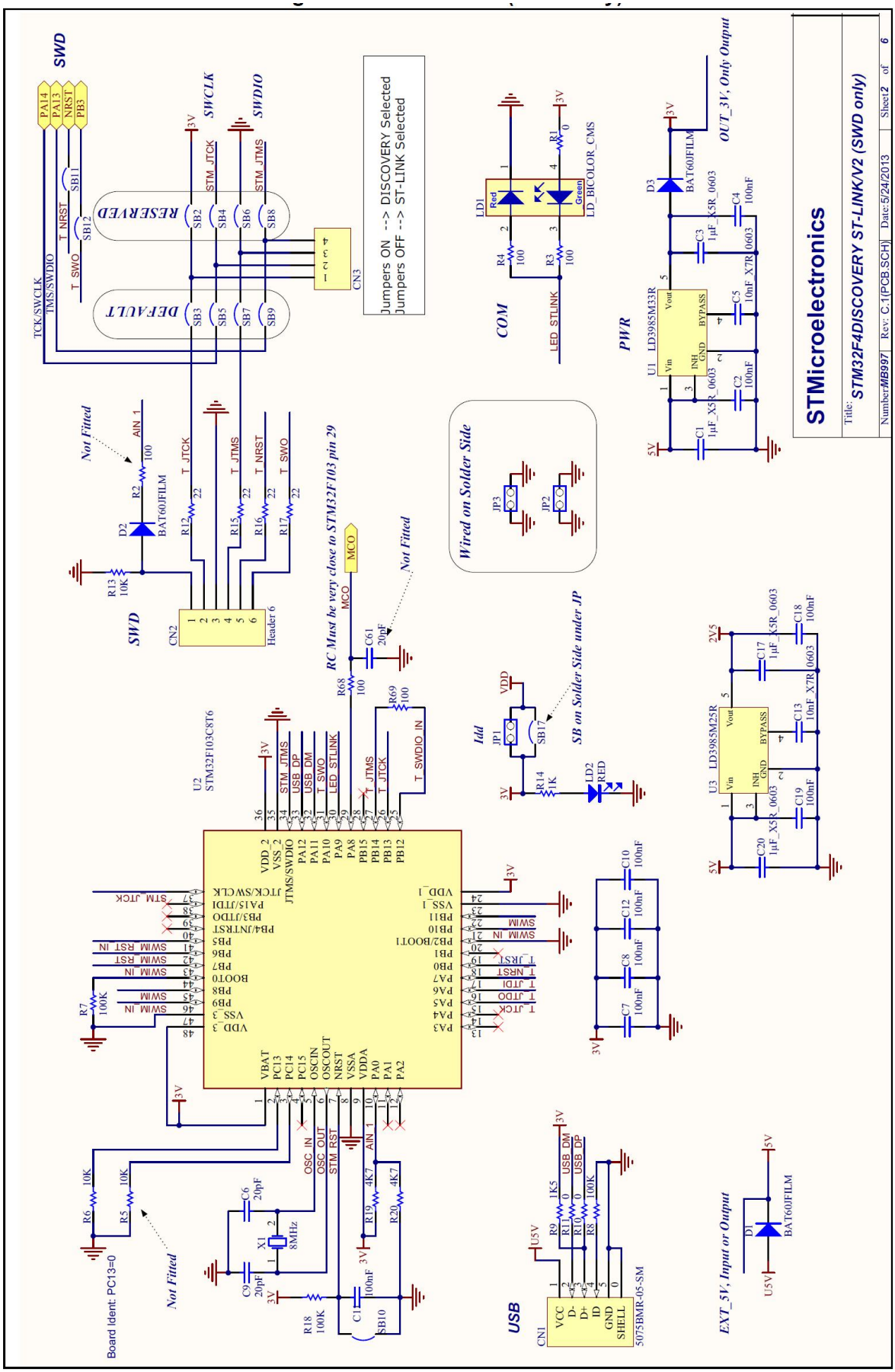
*Рис. 2.24. Структурна схема та зовнішній вигляд **STM32F4DISCOVERY***

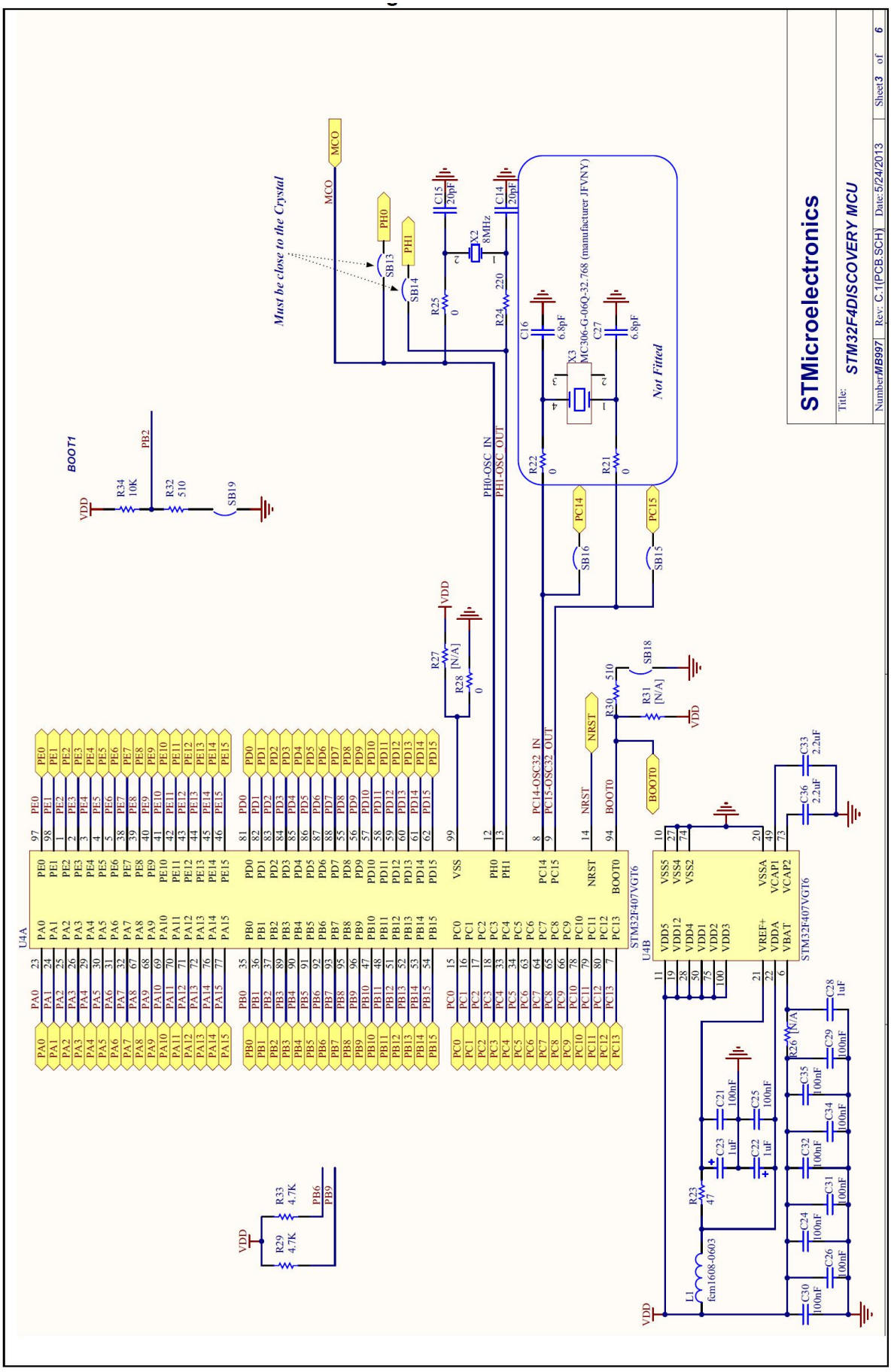
Принципова схема забражена на рис. х-у. На платі **STM32F4DISCOVERY** розміщено модуль відлагодження та програмування пам'яті на кристалі мікроконтролера, який використовує для зв'язку з мікроконтролером інтерфейс SWD, а до комп'ютера підключається через роз'єм міні-USB. Крім цього на платі розміщено роз'єм мікро-USB для використання порту USB-користувача в прикладних цілях. Також на платі розміщено вузол

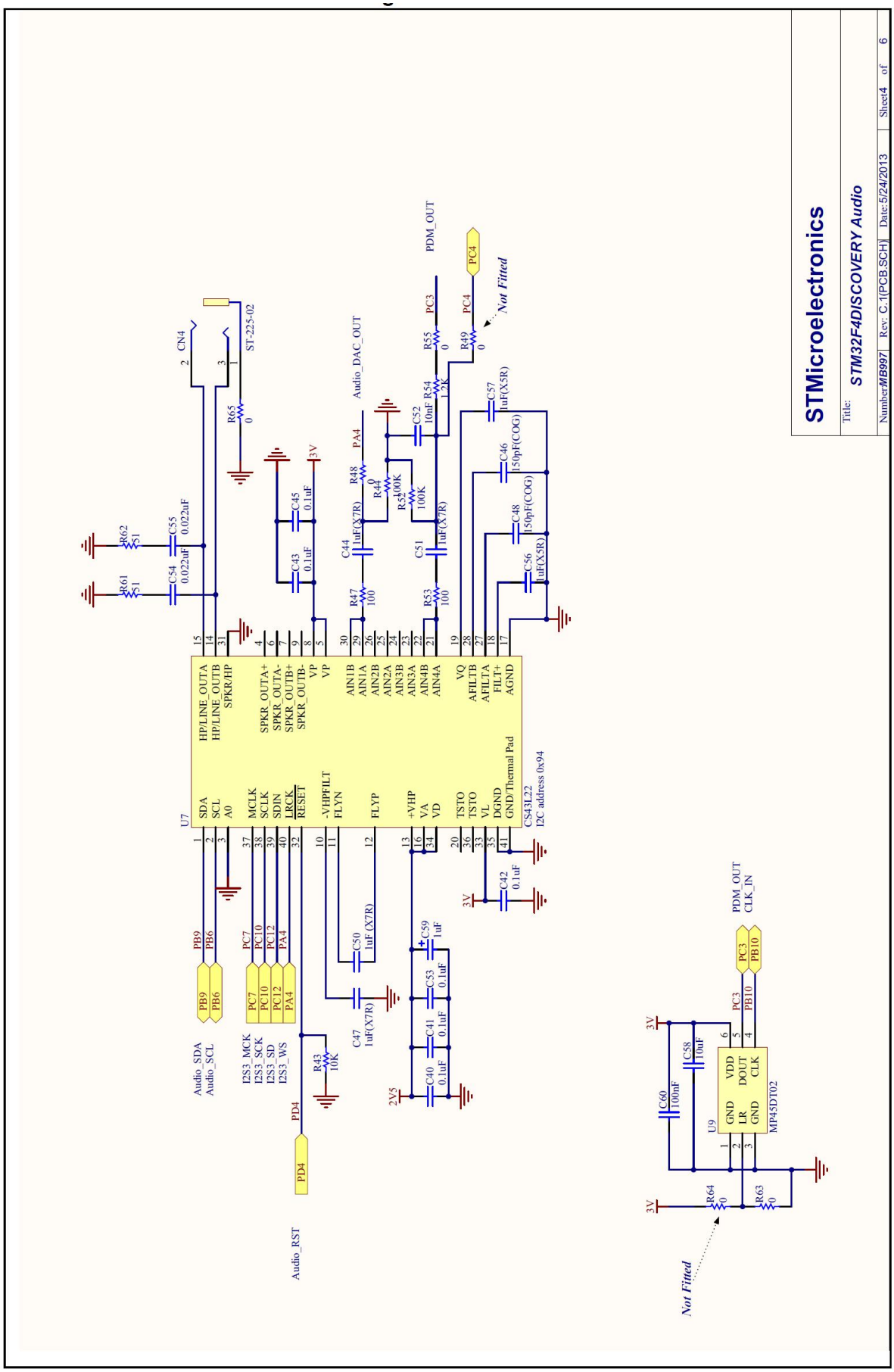
трьохкоординатного лінійного акселерометра (сенсор руху), вузол цифро-аналогового перетворювача аудіо DAC (CS43L22) з виведенням звукового сигналу через аудіо міні-роз'єм, ряд керованих кольорових світлодіодів, кнопка “Reset” та кнопка користувача “User”. Лінії мікроконтролера **STM32F407VGT6** виведені на зовнішні контакти плати, що дозволяє підключати зовнішні компоненти для відлагодження окремих вузлів МПС.

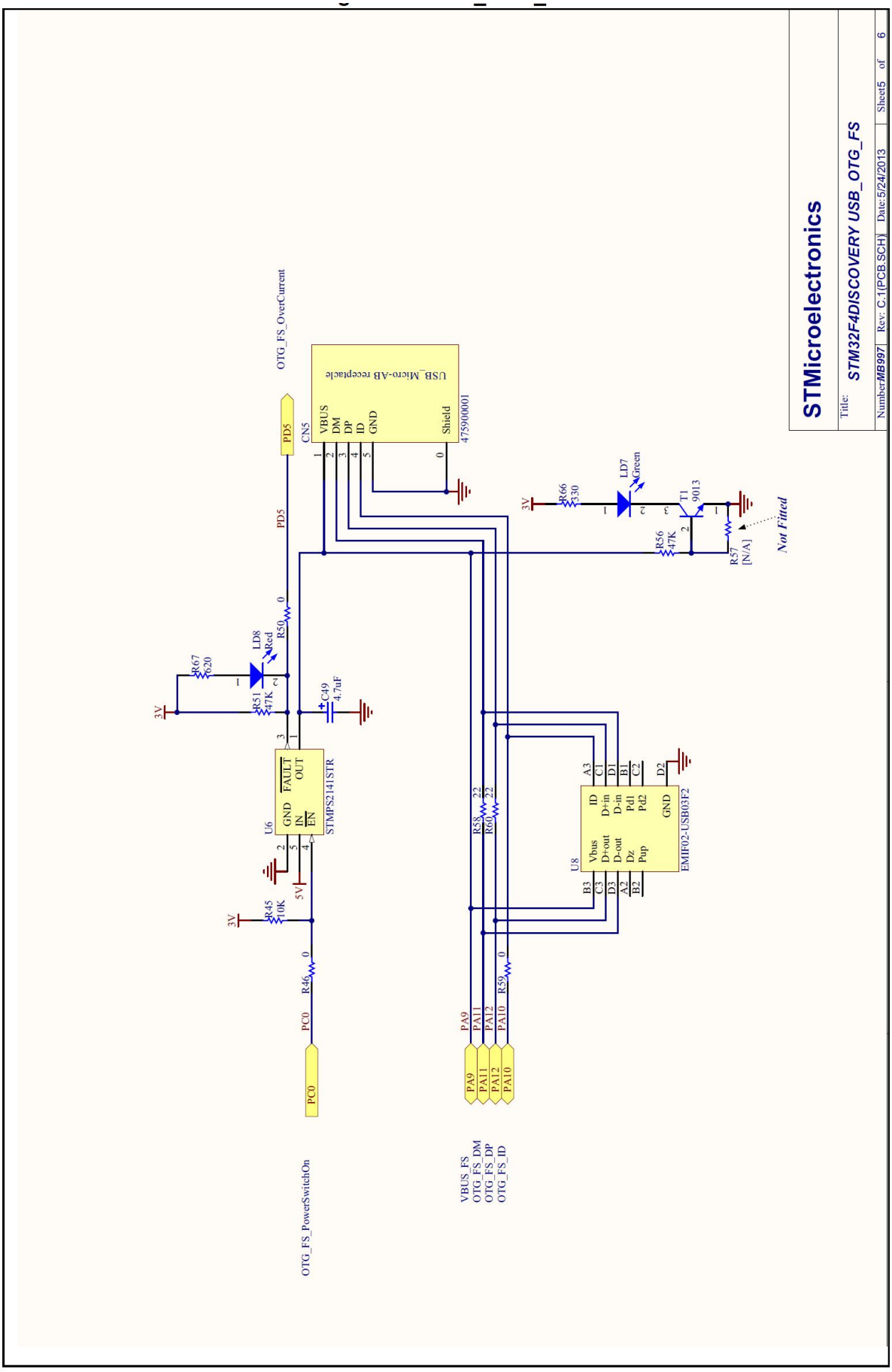
Нижче показано схеми модуля **STM32F4DISCOVERY** [2.3]**.**

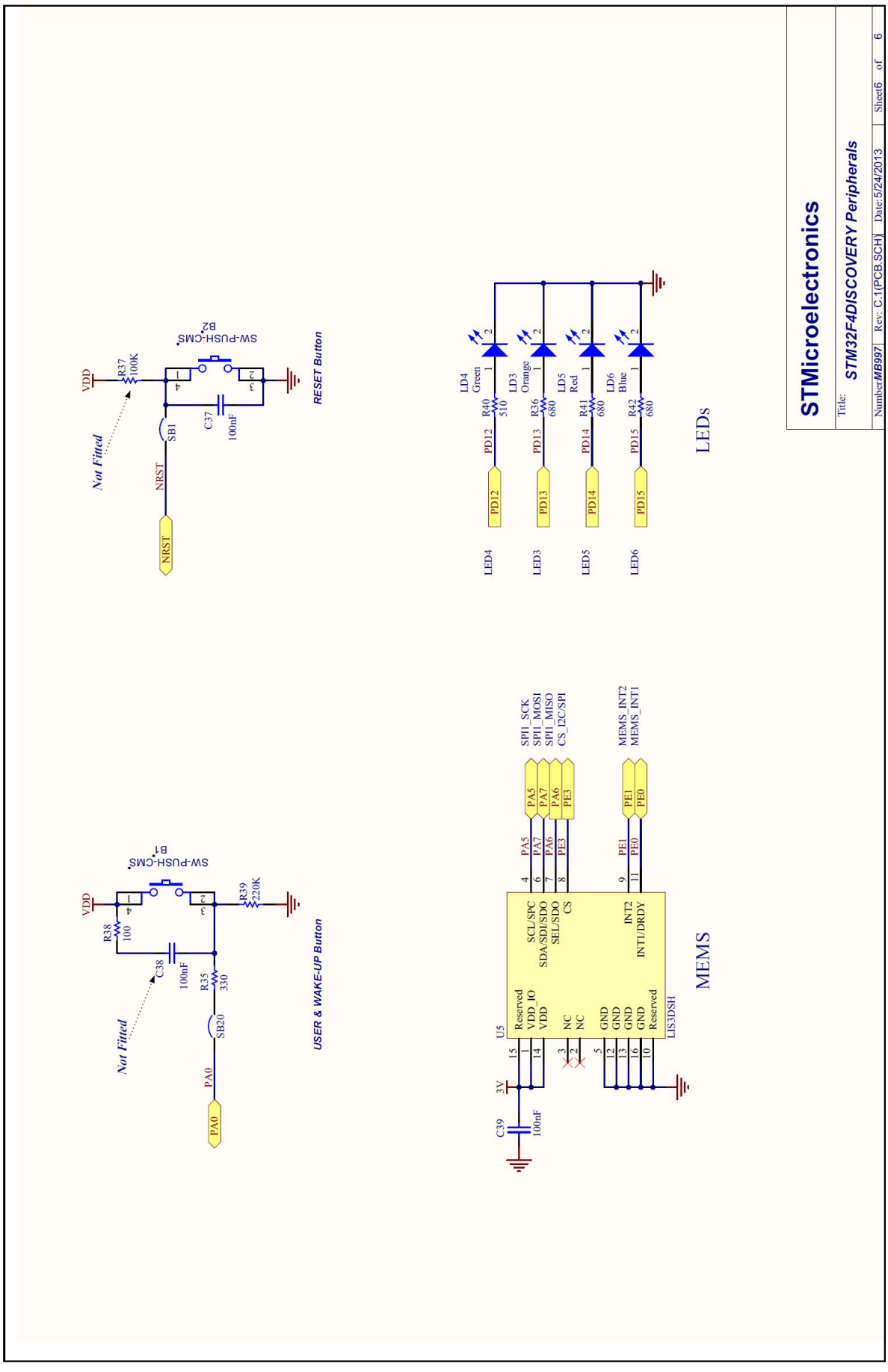



STMicroelectronics
Title: STM32F4DISCOVERY ST-LINK/V2 (SWD only)
Number:MB997 | Rev: C.1(PCB.SCH) | Date:5/24/2013 | Sheet2 of 6
SWD
SWCLK
SWDIO
RESERVED
DEFAULT
Jumpers ON --> DISCOVERY Selected
Jumpers OFF --> ST-LINK Selected
COM
PWR
OUT_3V, Only Output
USB
EXT_5V, Input or Output
Idd
Wired on Solder Side
RC Must be very close to STM32F103 pin 29
SB on Solder Side under JP
Not Fitted
Board Ident: PC13=0
STM32F103C8T6
LD3985M33R
BAT60JFILM


STMicroelectronics
Title: STM32F4DISCOVERY MCU
Number MB997 | Rev: C.1(PCB.SCH) | Date: 5/24/2013 | Sheet 3 of 6
Must be close to the Crystal
Not Fitted
MC306-G-06Q-32.768 (manufacturer JFVNY)
8MHz
BOOT1
PH0-OSC_IN
PH1-OSC_OUT
PC14-OSC32_IN
PC15-OSC32_OUT
STM32F407VGT6
U4A
U4B
MCO
NRST
BOOT0
VDD
PB2
PB6
PB9
R34 10K
R32 510
R33 4.7K
R29 4.7K
R25 0
R24 220
R22 0
R21 0
R27 [N/A]
R28 0
R30 510
R31 [N/A]
R23 47
R26 [N/A]
C15 20pF
C14 20pF
C16 6.8pF
C27 6.8pF
C21 100nF
C25 100nF
C23 1uF
C22 1uF
C33 2.2uF
C36 2.2uF
C28 1uF
C29 100nF
C35 100nF
C34 100nF
C32 100nF
C31 100nF
C24 100nF
C26 100nF
C30 100nF
L1 fcm1608-0603
SB13
SB14
SB15
SB16
SB18
SB19


STMicroelectronics
Title: STM32F4DISCOVERY Audio
Number MB997 Rev: C.1(PCB.SCH) Date: 5/24/2013 Sheet 4 of 6
U7
CS43L22
I2C address 0x94
U9
MP45DT02
CN4
ST-225-02
Audio_SDA
Audio_SCL
Audio_RST
I2S3_MCK
I2S3_SCK
I2S3_SD
I2S3_WS
PDM_OUT
CLK_IN
Audio_DAC_OUT
Not Fitted
Not Fitted


STMicroelectronics
Title:
STM32F4DISCOVERY USB_OTG_FS
Number:MB997 Rev: C.1(PCB.SCH) Date:5/24/2013 Sheet5 of 6
USB_Micro-AB receptacle
CN5
VBUS
DM
DP
ID
GND
Shield
47590001
OTG_FS_OverCurrent
PD5
LD8
Red
R67
620
R51
47K
R50
0
C49
4.7uF
3V
U6
STMPS2141STR
GND
FAULT
OUT
IN
EN
5V
R45
10K
R46
0
PC0
OTG_FS_PowerSwitchOn
R58
22
R60
22
R59
0
PA9
PA11
PA12
PA10
VBUS_FS
OTG_FS_DM
OTG_FS_DP
OTG_FS_ID
U8
EMIF02-USB03F2
Vbus
D+out
D-out
Dz
Pup
ID
D+in
D-in
Pd1
Pd2
GND
LD7
Green
R66
330
T1
9013
R56
47K
R57
[N/A]
Not Fitted


STMicroelectronics
Title: STM32F4DISCOVERY Peripherals
Number:MB997 Rev: C.1(PCB.SCH) Date: 5/24/2013 Sheet6 of 6
RESET Button
SW-PUSH-CMS
B2
VDD
R37
100K
Not Fitted
C37
100nF
SB1
NRST
NRST
LEDs
LD4
Green
LD3
Orange
LD5
Red
LD6
Blue
R40
510
R36
680
R41
680
R42
680
PD12
PD13
PD14
PD15
LED4
LED3
LED5
LED6
USER & WAKE-UP Button
SW-PUSH-CMS
B1
R38
100
C38
100nF
R35
330
SB20
PA0
R39
220K
MEMS
U5
LIS3DSH
Reserved
VDD_IO
VDD
NC
NC
GND
GND
GND
GND
Reserved
SCL/SPC
SDA/SDI/SDO
SEL/SDO
CS
INT2
INT1/DRDY
3V
C39
100nF
SPI1_SCK
SPI1_MOSI
SPI1_MISO
CS_I2C/SPI
MEMS_INT2
MEMS_INT1
PA5
PA7
PA6
PE3
PE1
PE0

4. Мікроконтролер STM32F407VGT6 з ядром Cortex-M4F

4.1. Ядро Cortex-M4F розроблено для проєктування вбудованих мікрокомп'ютерних систем середньої продуктивності з низькою споживною потужністю. Архітектурно це 32-розрядне RISC-ядро типу ARM нового покоління з повним набором інструкцій ARM, реалізацією інструкцій DSP, вбудованим процесором плаваючої коми (FPU) та максимальною робочою частотою до 200 MHz. Ядро має вбудований модуль захисту пам'яті який підвищує безпеку роботи виконання програм. Для оптимального програмування FPU використовується спеціальна мета-мова. Все це дозволяє ефективно обробляти сигнали та реалізовувати складні алгоритми керування.

На базі ядра Cortex-M4F різні фірми освоїли випуск широкої номенклатури мікроконтролерів з різними об'ємами інтегрованої на кристалі пам'яті та різноманітною номенклатурою інтегрованих периферійних пристроїв. Одними з таких мікроконтролерів є мікроконтролери серії STM32F4xx фірми STMicroelectronics які стали досить популярними серед розробників.

Вказані мікроконтролери характеризуються наявністю інтегрованої швидкодіючої пам'яті Flash до 1 Mb та наявністю кеш-пам'яті що забезпечує звертання до пам'яті на максимальній частоті процесора без циклів очікування. Система обробки подій в реальному часі не вимагає втручання процесора що забезпечує обробку без затримок. Номенклатура периферійних пристроїв різних мікроконтролерів поряд з класичними інтерфейсними пристроями включає такі комунікаційні інтерфейси як Ethernet MAC з підтримкою стандарту IEEE1588, CAN, USB, інтерфейси для підключення відеокамери та графічних дисплеїв, 12 та 16-розрядні багатоканальні швидкодіючі аналого-цифрові перетворювачі (АЦП), спеціалізовані цифро-аналогові перетворювачі (ЦАП). Для забезпечення високої продуктивності при обміні даними на кристалах інтегровані канали прямого доступу до пам'яті. Деякі мікроконтролери мають інтегровані на кристалі такі специфічні вузли як криптографічний процесор, модуль обчислення контрольних сум CRC, генератор випадкових чисел.

Важливою особливістю вказаних мікроконтролерів є можливість функціонування в режимі пониженого споживання при батарейному живленні. Також при пропаданні основного живлення забезпечується зберігання важливих даних в SRAM з батарейним живленням та функціонування таймера реального часу. Мікроконтролери випускаються в різних корпусах з різною кількістю виводів від 64 до 176, що дозволяє оптимізувати апаратні засоби при різних застосуваннях. Напруга живлення мікроконтролерів від 1.8V до 3.6V , температура в робочому режимі від - 40 до +105 градусів С.

4.2.Внутрішня структура мікроконтролера STM32F407VGT6.

Основні характеристики

• ядро 32 розряди;

• максимальна частота ядра 168 MHz;

• вбудований процесор плаваючої коми FPU;

• інструкції ARM та DSP;

• вбудована Flash пам'ять програм до 1Mb та SRAM даних до 192 Kb з можливістю зовнішнього розширення;

• SRAM 4Kb з зовн. резервним живленням;

• широка номенклатура інтегрованої периферії;

• VDD = 3.3 V (1.8 V ≤ VDD ≤ 3.6 V)

Вузли STM32F40x

• ядро ARM® Cortex™-M4F 168 MHz;
• система синхронізації PLL;
• інтегрований процесор FPU;
• інтегрована Flash пам’ять програм до 1Mb;
• інтегрована SRAM даних до 192 Kb;
• інтегрована резервна SRAM даних 4Kb (Vbat);
• адаптивний акселератор памя’ті (ART Accelerator);
• блок захисту пам’яті (MPU);
• модуль обчислення контрольних сумм (CRC unit);
• генератор випадкових чисел (RNG);
• мульти-AHB матриця шин 32 р;
• контролер прямого доступу до пам’яті (DMA);
• контролер статичної пам’яті (FSMC) з можливісю конфігурації керування РКІ;
• контролер вкладених векторних переривань (NVIC);
• контролер зовнішніх переривань (EXTI);
• широка номенклатура інтерфейсних вузлів для паралельного та послідовного обміну (GPIO, (SDIO)-SD/SDIO/MMC, USART, SPI, I²C, CAN, USB, Ethernet, I2S);
• інтерфейс цифрової камери (DCMI);
• інтегровані 12р АЦП та ЦАП;
• універсальні таймери, RTC, WDT, SysTick ;
• вбудовані макро-регістри трасування (Embedded Trace Macrocell, ETM);
• JTAG та SW інтерфейси для трасування та програмування інтегрованої Flash;
• інтегрований стабілізатор напруги живлення;
• вузол керування живленням;

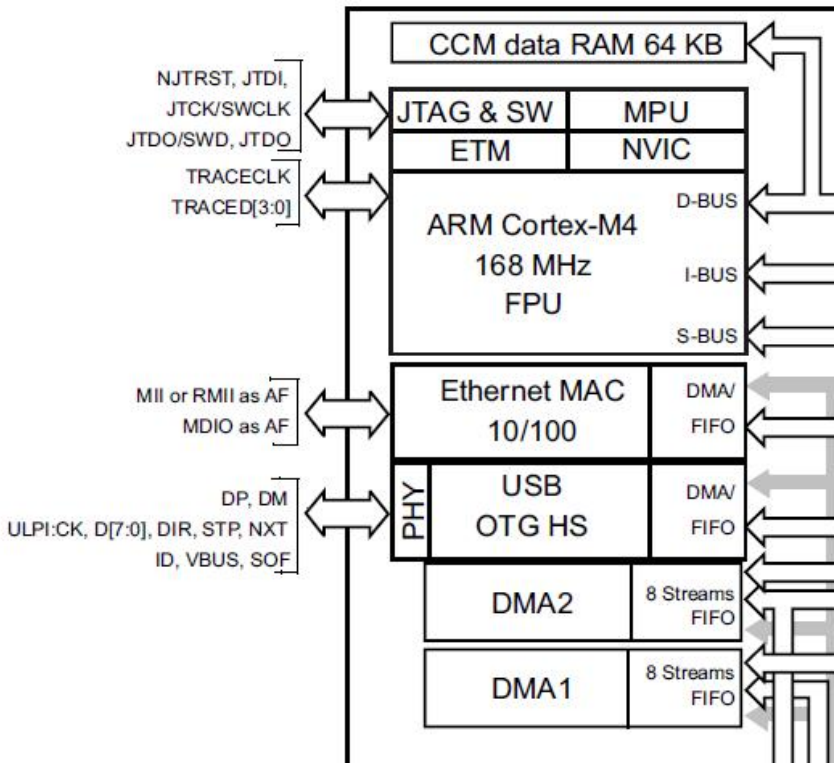


Рис. 2.25а. Внутрішня структура STM32F407

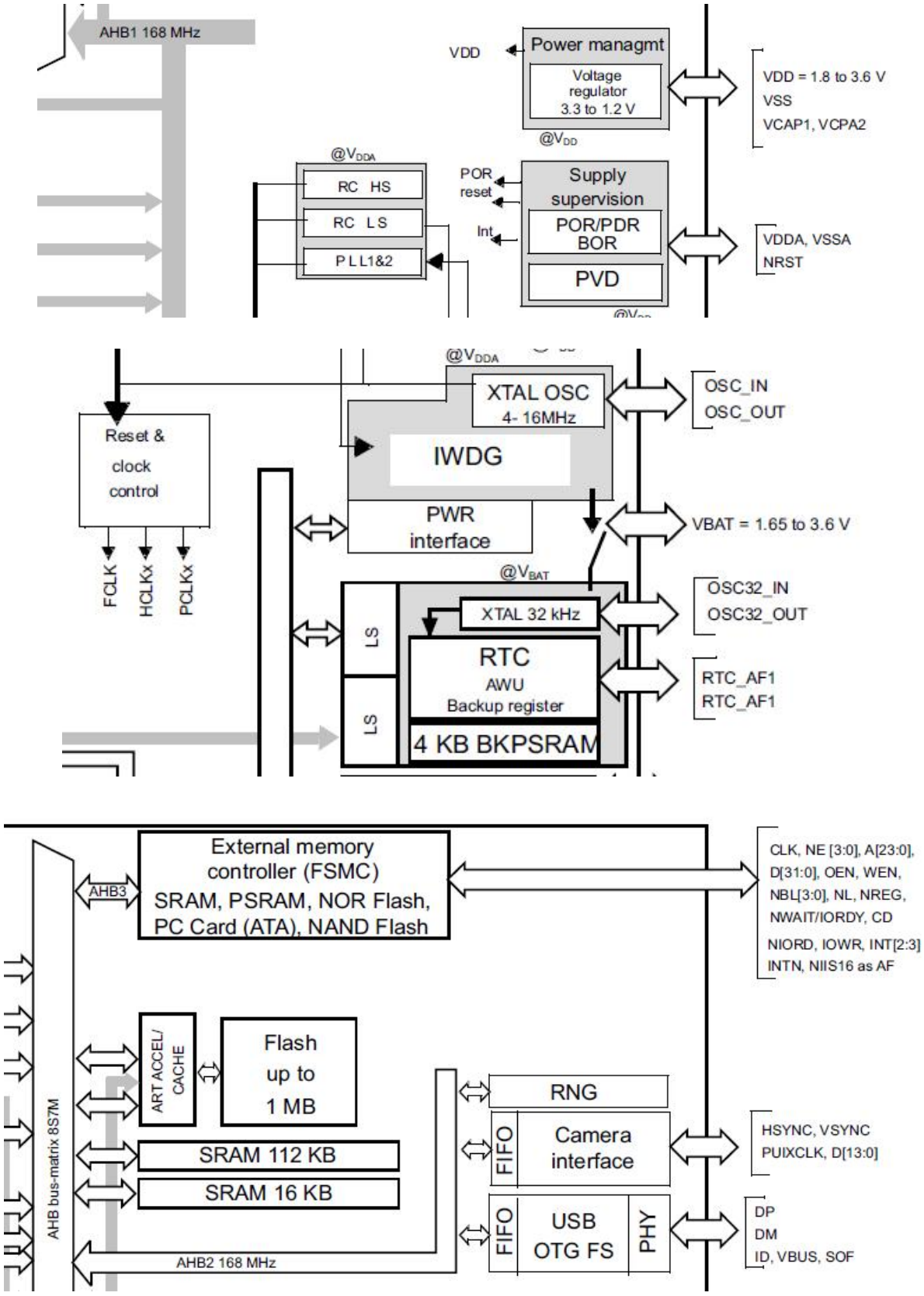


Рис. 2.25б. Внутрішня структура STM32F407

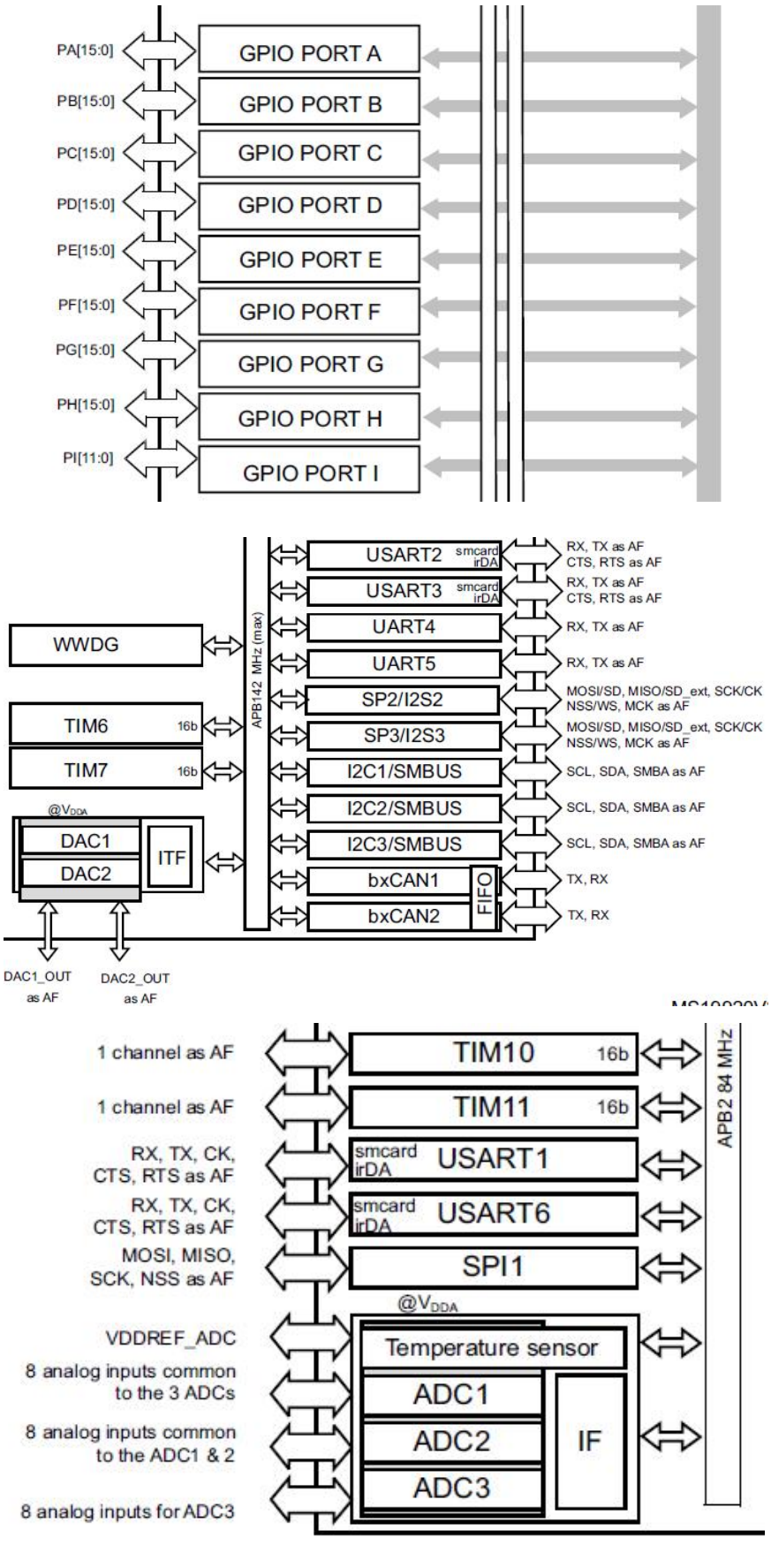


Рис. 2.25в. Внутрішня структура STM32F407

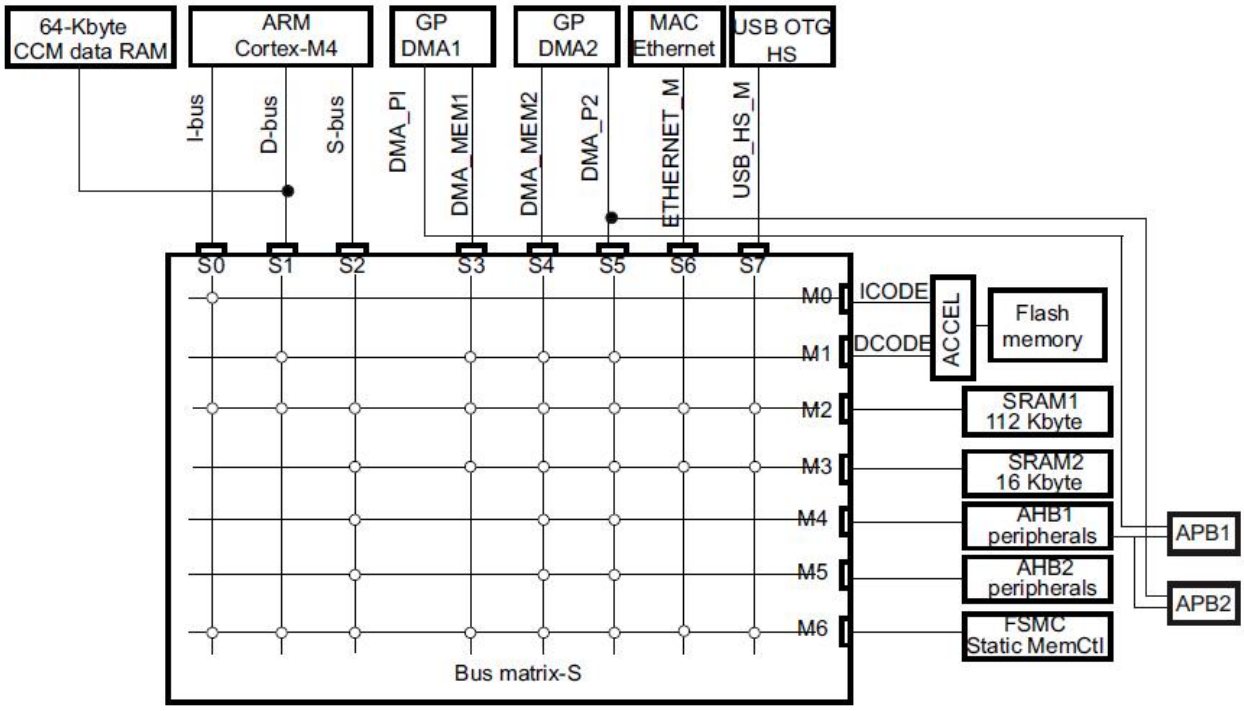


Рис. 2.26. Мульти-AHB матриця шин 32 р

КПДП (DMA)
memory-to-memory, peripheral-to-memory and
memory-to-peripheral

- SPI and I2S
- I2C
- USART
- General-purpose, basic and advanced-control timers TIMx
- DAC
- SDIO
- Camera interface (DCMI)
- ADC.

Живлення

• VDD = 1.8 to 3.6 V: external power supply for I/Os and the internal regulator (when enabled), provided externally through VDD pins.

• VSSA, VDDA = 1.8 to 3.6 V: external analog power supplies for ADC, DAC, Reset blocks, RCs and PLL. VDDA and VSSA must be connected to VDD and VSS, respectively.

• VBAT = 1.65 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when VDD is not present.

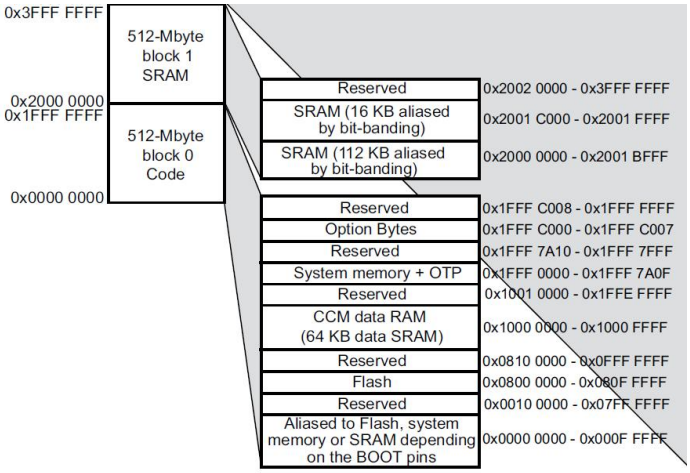


Рис. 2.27а. ***Структура пам'яті STM32F40x***

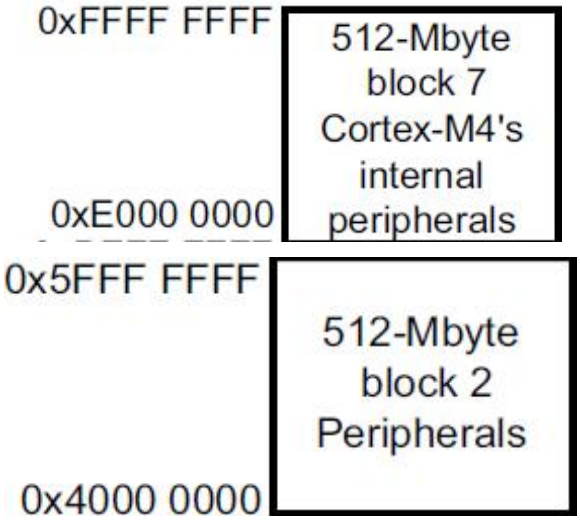


Рис. 2.27б. ***Структура пам'яті STM32F40x***

Bus	Boundary address	Peripheral
	0xE00F FFFF - 0xFFFF FFFF	Reserved
Cortex-M4	0xE000 0000 - 0xE00F FFFF	Cortex-M4 internal peripherals
	0xA000 1000 - 0xDFFF FFFF	Reserved
AHB3	0xA000 0000 - 0xA000 0FFF	FSMC control register
	0x9000 0000 - 0x9FFF FFFF	FSMC bank 4
	0x8000 0000 - 0x8FFF FFFF	FSMC bank 3
	0x7000 0000 - 0x7FFF FFFF	FSMC bank 2
	0x6000 0000 - 0x6FFF FFFF	FSMC bank 1
	0x5006 0C00- 0x5FFF FFFF	Reserved
AHB2	0x5006 0800 - 0x5006 0BFF	RNG
	0x5005 0400 - 0x5006 07FF	Reserved
	0x5005 0000 - 0x5005 03FF	DCMI
	0x5004 0000- 0x5004 FFFF	Reserved
	0x5000 0000 - 0x5003 FFFF	USB OTG FS
	0x4008 0000- 0x4FFF FFFF	Reserved

Рис. 2.28а. Регістри STM32F40x

Bus	Boundary address	Peripheral
AHB1	0x4004 0000 - 0x4007 FFFF	USB OTG HS
	0x4002 9400 - 0x4003 FFFF	Reserved
	0x4002 9000 - 0x4002 93FF	ETHERNET MAC
	0x4002 8C00 - 0x4002 8FFF	
	0x4002 8800 - 0x4002 8BFF	
	0x4002 8400 - 0x4002 87FF	
	0x4002 8000 - 0x4002 83FF	
	0x4002 6800 - 0x4002 7FFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 5FFF	Reserved
	0x4002 4000 - 0x4002 4FFF	BKPSRAM
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2400 - 0x4002 2FFF	Reserved
	0x4002 2000 - 0x4002 23FF	GPIOI
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA
	0x4001 5800- 0x4001 FFFF	Reserved

Рис. 2.28б. Регістри STM32F40x

Регістри паралельних портів

GPIO Registers

Configuration Registers

STM32 містить чотири регістри конфігурації для кожного з портів:

- Port mode register – GPIOx_MODER
- Output type register – GPIOx_OTYPER
- Speed register – GPIOx_OSPEEDR
- Pull-up/Pull-down register – GPIOx_PUPDR

Кожен з цих регістрів має довжину 32 біти, хоча і не всі біти використовуються у всіх регістрах.

Port Mode Register – GPIOx_MODER

Цей 32-розрядний регістр має 2 бітні значення, які визначають режим роботи. Можна встановити такі режими:

Bit Values	Description
00	Input
01	General purpose output

10	Alternate function
11	Analog

Port	**Reset Value**
A	0xA800 0000
B	0x0000 0280
All others	0x0000 0000

Output Type Register – GPIOx_OTYPER

Bit Value	**Description**
0	Push/pull output
1	Open drain output

Speed Register – GPIOx_OSPEEDR

Bit Values	**Description**
00	2 MHz Low speed
01	25 MHz Medium speed
10	50 MHz Fast speed
11	100 MHz High speed on 30pF 80 MHz High speed on 15 pF

Pull-up/Pull-down register – GPIOx_PUPDR

Bit Values	**Description**
00	No pull-up or pull-down resistor
01	Pull-up resistor
10	Pull-down resistor
11	Reserved

Input data register – GPIOx_IDR

Bit Value	**Description**
0	High logic value
1	Low logic value

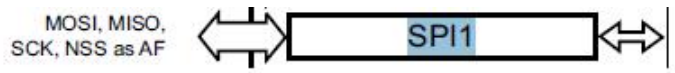
Output data register – GPIOx_ODR

Bit Value	**Description**
0	High logic value
1	Low logic value

4.3. **Інтерфейс SPI мікроконтролера STM32F407VGT6**

Кольоровий графічний індикатор **ILI9341** до відлагоджувального модуля **STM32F4 DISCOVERY** підключений через інтерфейс **SPI1** мікроконтролера **STM32F407VGT6.**

SPI1 працює в режимі **master.**



PA4	I/O	TTa	(4)	SPI1_NSS / SPI3_NSS / USART2_CK / DCMI_HSYNC / OTG_HS_SOF/ I2S3_WS/ EVENTOUT
PA5	I/O	TTa	(4)	SPI1_SCK/ OTG_HS_ULPI_CK / TIM2_CH1_ETR/ TIM8_CH1N/ EVENTOUT
PA6	I/O	FT	(4)	SPI1_MISO / TIM8_BKIN/TIM13_CH1 / DCMI_PIXCLK / TIM3_CH1 / TIM1_BKIN/ EVENTOUT
PA7	I/O	FT	(4)	SPI1_MOSI/ TIM8_CH1N / TIM14_CH1/TIM3_CH2/ ETH_MII_RX_DV / TIM1_CH1N / ETH_RMII_CRS_DV/ EVENTOUT

*Рис. 2.29 . Інтерфейс SPI1 мікроконтролера **STM32F407VGT6***

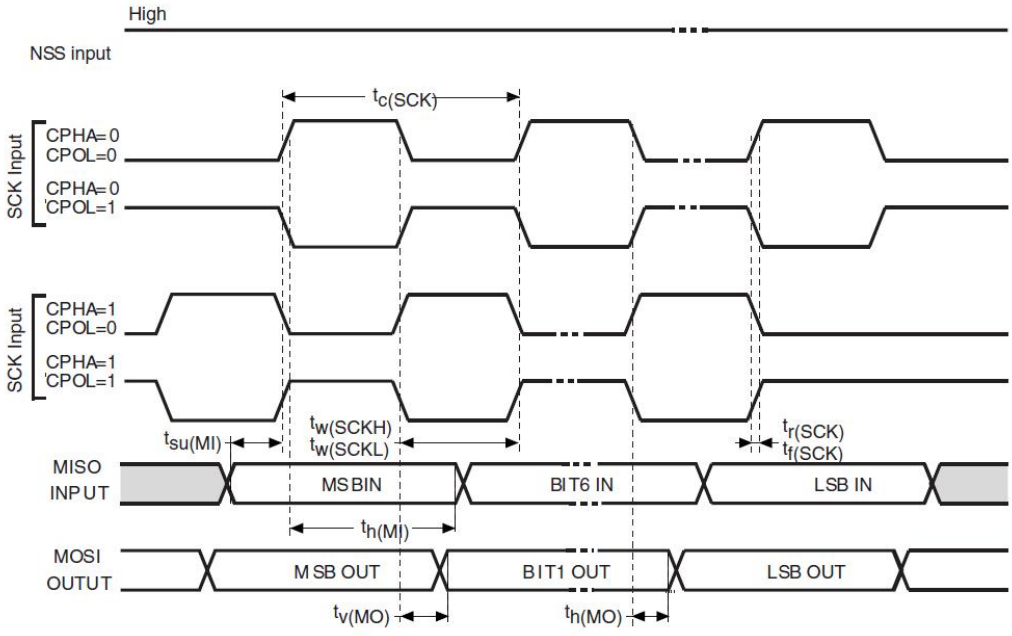


Рис. 2.30. Часові діаграми SPI в режимі master

4. Кольоровий графічний індикатор типу ILI9341

Кольоровий графічний індикатор TFT 2.8”, 240×320 пікселів з інтерфейсом SPI на контролері ILI9341 з резистивною сенсорною панеллю і контролером XPT2046 та слотом для SD-карти.

Рис. 2.31. Кольоровий графічний індикатор ILI9341. Вид з фронтальної сторони

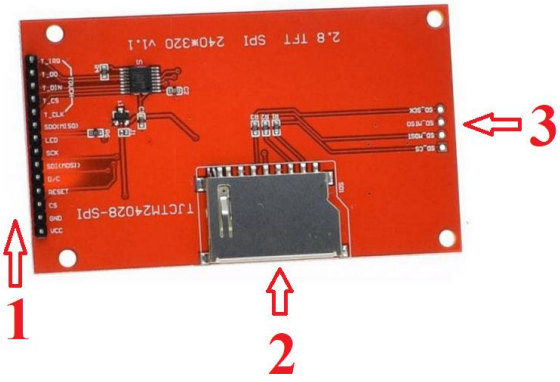
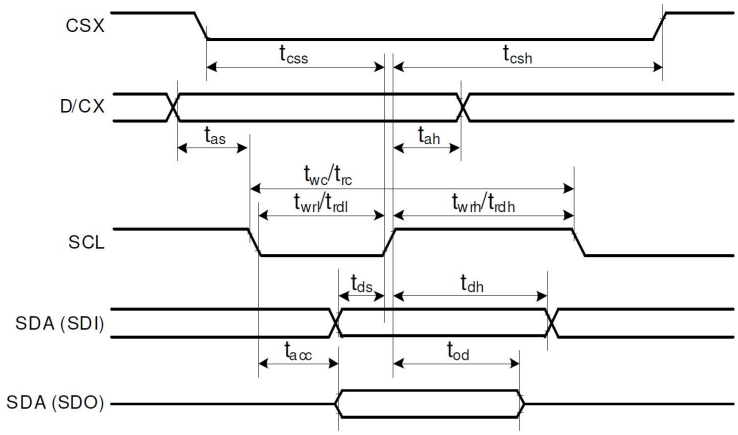


Рис. 2.32. Кольоровий графічний індикатор ILI9341. Вид з тильної сторони

1-інтерфейс керування дисплеєм та сенсорною панеллю;
2-слот SD-картки;
3-інтерфейс SD-картки



Signal	Symbol	Parameter	min	max	Unit	Description
CSX	tcss	Chip select time (Write)	40	-	ns	
	tcsh	Chip select hold time (Read)	40	-	ns	
SCL	twc	Serial clock cycle (Write)	100	-	ns	
	twrh	SCL “H” pulse width (Write)	40	-	ns	
	twrl	SCL “L” pulse width (Write)	40	-	ns	
	trc	Serial clock cycle (Read)	150	-	ns	
	trdh	SCL “H” pulse width (Read)	60	-	ns	
	trdl	SCL “L” pulse width (Read)	60	-	ns	
D/CX	tas	D/CX setup time	10	-		
	tah	D/CX hold time (Write / Read)	10	-		
SDA / SDI (Input)	tds	Data setup time (Write)	30	-	ns	
	tdh	Data hold time (Write)	30	-	ns	
SDA / SDO (Output)	tacc	Access time (Read)	10	-	ns	For maximum CL=30pF
	tod	Output disable time (Read)	10	50	ns	For minimum CL=8pF

Рис. 2.33. Кольоровий графічний індикатор ILI9341. Часові діаграми SPI дисплея

Програмні середовища для розроблення програмного забезпечення

STM32Cube

STM32Cube – це набір **free** програмних пакетів під Windows для STM32 мікроконтролерів та мікропроцесорів (далі компонентів STM32) фірми STMicroelectronics. STM32Cube може використовуватися користувачами, які вже використовуть IDE типу Keil або iAR, в які можуть легко інтегруватися різні компоненти, такі як STM32CubeMX, STM32CubeProgrammer або STM32CubeMonitor.

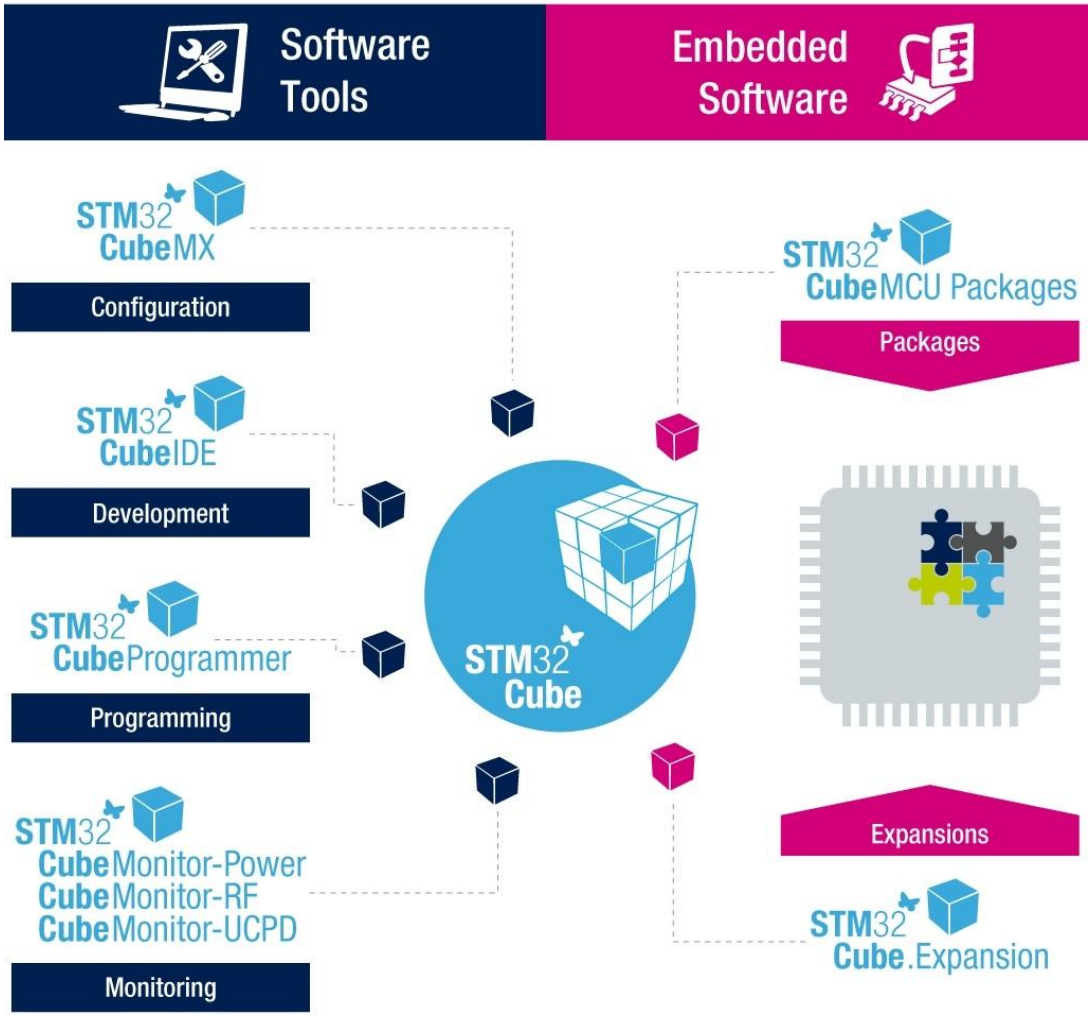


Рис. 2.34. Пакети STM32Cube

STM32CubeMX – програмний пакет для конфігурації компонентів STM32. Дозволяє генерувати програмний код на C для компонентів з ядром Cortex-M та для пристроїв з ядром Cortex-A під Linux. Пакет має простий але достатньо розвинутий графічний інтерфейс, що полегшує його використання.

STM32CubeIDE – інтегроване середовище на основі рішень з відкритим кодом, таких як Eclipse або GNU C / C ++. Забезпечує компіляцію та розширені функції відлагодження.

STM32CubeProgrammer – програматор машинного коду в програмну пам'ять компонентів STM32. Забезпечує читання, запис та контрольну перевірку машинного коду через різноманітні інтерфейси: JTAG, SWD, UART, USB DFU, I2C, SPI, CAN тощо.

STM32CubeMonitor – інструменти моніторингу, які допомагають розробникам налаштувати функціонування та продуктивність своїх додатків у режимі реального часу.

STM32CubeMX

Рис. 2.35. Вікно запуску STM32CubeMX

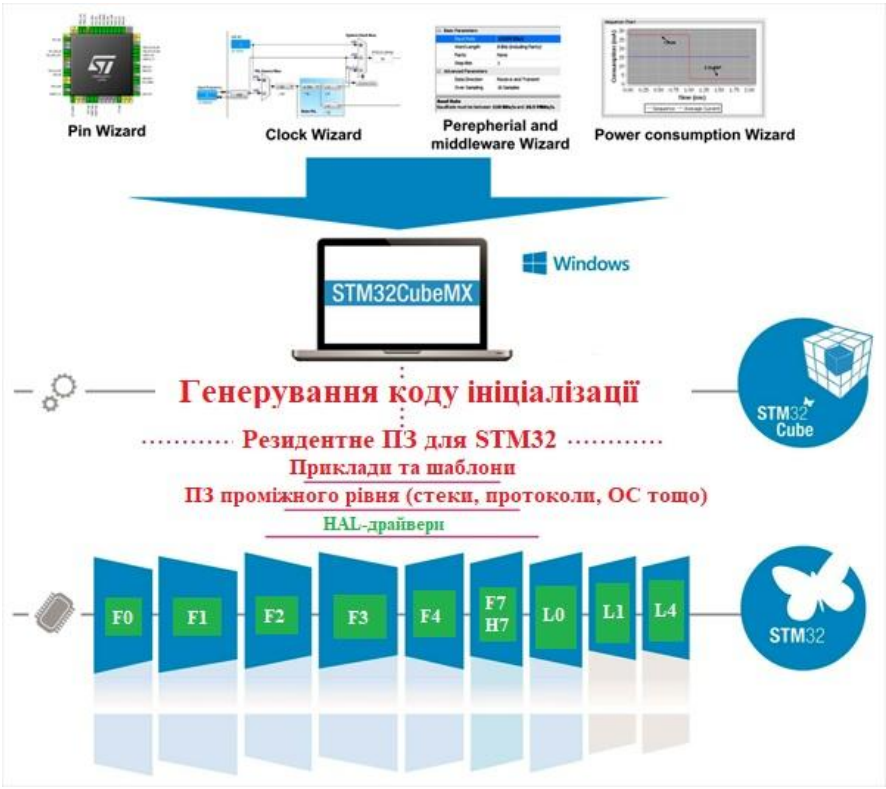
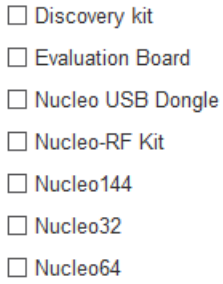


Рис. 2.36. Пакет STM32CubeMX

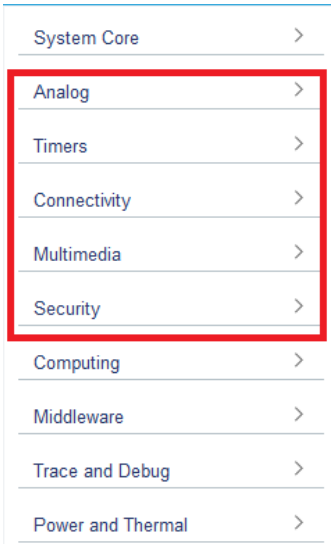
STM32CubeMX на основі графічного інтерфейсу забезпечує:

– вибір конкретного мікропроцесорного компонента STM32 з відповідним ядром: **F0-L4**;

– вибір типу **STBoard** при використанні стандартних апаратних засобів налагодження:



– вибір та конфігурацію інтегрованих на кристалі вузлів для використання в мікропроцесорній системі відповідного функціонального призначення (відмічено червоним кольором) та інших вузлів мікропроцесорного компонента STM32:



– ініціалізацію системи синхронізації на основі вибраного апаратного базису (зовнішній сигнал синхронізації, тактовий генератор на базі кварцового резонатора відповідної частоти);

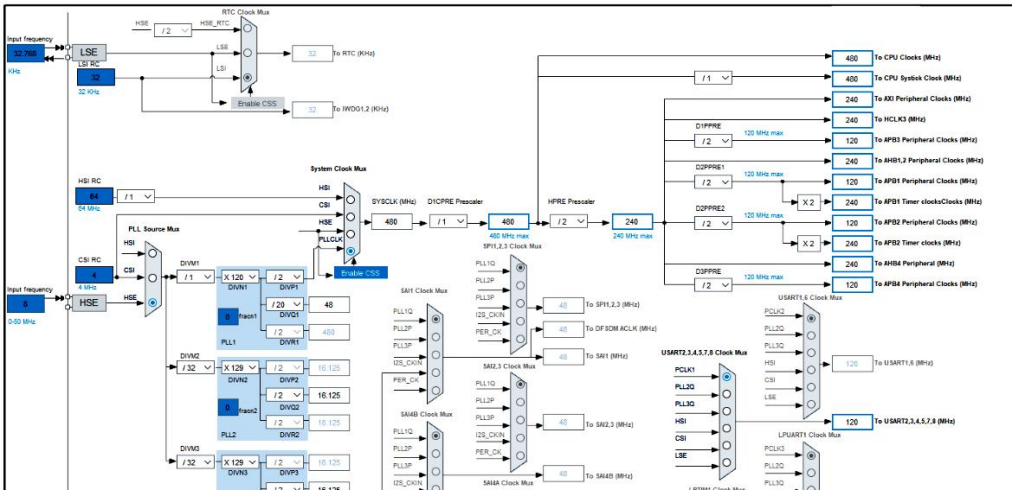


Рис. 2.37. Ініціалізація системи синхронізації

- конфігурацію основних ліній мікропроцесорного компонента STM32 у відповідності до вибраних вузлів для мікропроцесорної системи відповідного функціонального призначення:

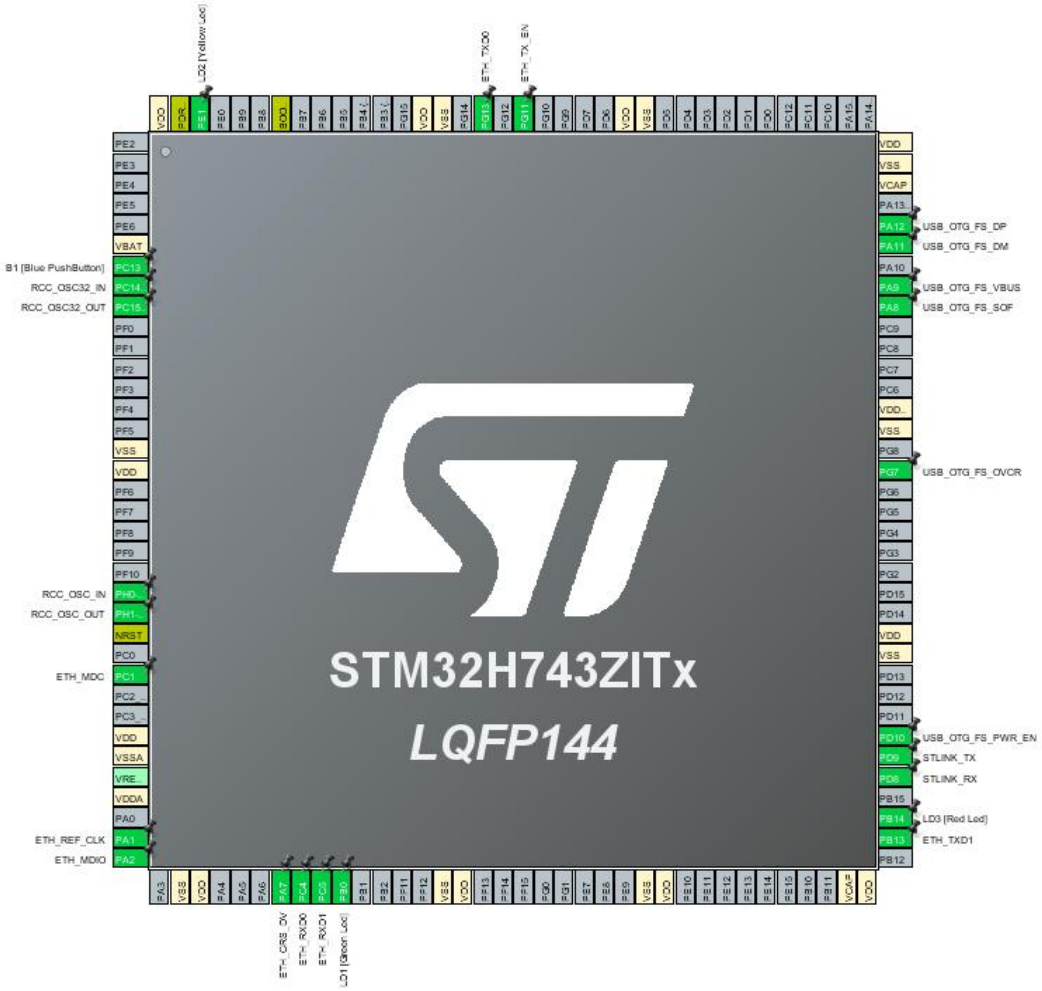


Рис. 2.38. Конфігурація основних ліній

– вибір середовища IDE, під яке STM32CubeMX згенерує код:

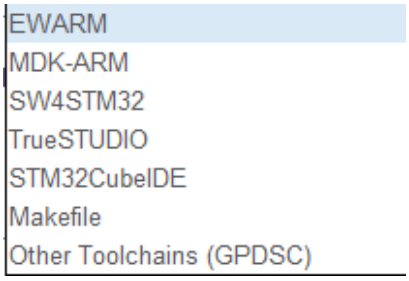


Рис. 2.39. Вибір середовища IDE

Література

2.1. RM0090 STM32F405xx_07xx Reference manual.pdf
2.2. STM32F407xx.pdf
2.3. STM32F4DISCOVERY User manual.pdf
2.4. Конспект лекцій з дисципліни “Мікропроцесорні системи”

Програмні модулі для керування та виводу інформації
на кольоровий графічний індикатор ILI9341

```
//tft_lcd.c

#include "tft_lcd.h"
#include "font.h"

extern const font_type TimesNewRoman;

void LCD_SendCommand(u8 com) {
  TFT_DC_RESET;
  TFT_CS_RESET;
  spi1_send(com);
  TFT_CS_SET;
}

void LCD_SendData(u8 data) {
  TFT_DC_SET;
  TFT_CS_RESET;
  spi1_send(data);
  TFT_CS_SET;
}

u8 LCD_Receve() {
  return 0;
}

void LCD_Init(){
  LCD_InitGPIO();

  TFT_CS_SET;
  spi1_init();


  TFT_RST_SET;
  LCD_SendCommand(ILI9341_RESET);
  Delay(100);

  /// commands here
  LCD_SendCommand(ILI9341_POWERA);
  LCD_SendData(0x39);
  LCD_SendData(0x2C);
  LCD_SendData(0x00);
  LCD_SendData(0x34);
  LCD_SendData(0x02);
  LCD_SendCommand(ILI9341_POWERB);
```

```
LCD_SendData(0x00);
LCD_SendData(0xC1);
LCD_SendData(0x30);
LCD_SendCommand(ILI9341_DTCA);
LCD_SendData(0x85);
LCD_SendData(0x00);
LCD_SendData(0x78);
LCD_SendCommand(ILI9341_DTCB);
LCD_SendData(0x00);
LCD_SendData(0x00);
LCD_SendCommand(ILI9341_POWER_SEQ);
LCD_SendData(0x64);
LCD_SendData(0x03);
LCD_SendData(0x12);
LCD_SendData(0x81);
LCD_SendCommand(ILI9341_PRC);
LCD_SendData(0x20);
LCD_SendCommand(ILI9341_POWER1);
LCD_SendData(0x23);
LCD_SendCommand(ILI9341_POWER2);
LCD_SendData(0x10);
LCD_SendCommand(ILI9341_VCOM1);
LCD_SendData(0x3E);
LCD_SendData(0x28);
LCD_SendCommand(ILI9341_VCOM2);
LCD_SendData(0x86);
LCD_SendCommand(ILI9341_MAC);
LCD_SendData(0x48);
LCD_SendCommand(ILI9341_PIXEL_FORMAT);
LCD_SendData(0x55);
LCD_SendCommand(ILI9341_FRC);
LCD_SendData(0x00);
LCD_SendData(0x18);
LCD_SendCommand(ILI9341_DFC);
LCD_SendData(0x08);
LCD_SendData(0x82);
LCD_SendData(0x27);
LCD_SendCommand(ILI9341_3GAMMA_EN);
LCD_SendData(0x00);
LCD_SendCommand(ILI9341_COLUMN_ADDR);
LCD_SendData(0x00);
LCD_SendData(0x00);
LCD_SendData(0x00);
LCD_SendData(0xEF);
LCD_SendCommand(ILI9341_PAGE_ADDR);
LCD_SendData(0x00);
LCD_SendData(0x00);
LCD_SendData(0x01);
LCD_SendData(0x3F);
```

```
  LCD_SendCommand(ILI9341_GAMMA);
  LCD_SendData(0x01);
  LCD_SendCommand(ILI9341_PGAMMA);
  LCD_SendData(0x0F);
  LCD_SendData(0x31);
  LCD_SendData(0x2B);
  LCD_SendData(0x0C);
  LCD_SendData(0x0E);
  LCD_SendData(0x08);
  LCD_SendData(0x4E);
  LCD_SendData(0xF1);
  LCD_SendData(0x37);
  LCD_SendData(0x07);
  LCD_SendData(0x10);
  LCD_SendData(0x03);
  LCD_SendData(0x0E);
  LCD_SendData(0x09);
  LCD_SendData(0x00);
  LCD_SendCommand(ILI9341_NGAMMA);
  LCD_SendData(0x00);
  LCD_SendData(0x0E);
  LCD_SendData(0x14);
  LCD_SendData(0x03);
  LCD_SendData(0x11);
  LCD_SendData(0x07);
  LCD_SendData(0x31);
  LCD_SendData(0xC1);
  LCD_SendData(0x48);
  LCD_SendData(0x08);
  LCD_SendData(0x0F);
  LCD_SendData(0x0C);
  LCD_SendData(0x31);
  LCD_SendData(0x36);
  LCD_SendData(0x0F);
  LCD_SendCommand(ILI9341_SLEEP_OUT);

  Delay(100);
  LCD_SendCommand(ILI9341_DISPLAY_ON);
  LCD_SendCommand(ILI9341_GRAM);
}

void LCD_InitGPIO() {
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
    GPIO_InitTypeDef gpio;
    GPIO_StructInit(&gpio);

    gpio.GPIO_Pin = TFT_DC_PIN | TFT_RESET_PIN | TFT_CS_PIN;
    gpio.GPIO_Mode = GPIO_Mode_OUT;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;
```

```
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_PuPd = GPIO_PuPd_DOWN;
    GPIO_Init(GPIOA,&gpio);
}

void LCD_SetCursorPosition(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2) {
        LCD_SendCommand(ILI9341_COLUMN_ADDR);
        LCD_SendData(x1 >> 8);
        LCD_SendData(x1 & 0xFF);
        LCD_SendData(x2 >> 8);
        LCD_SendData(x2 & 0xFF);

        LCD_SendCommand(ILI9341_PAGE_ADDR);
        LCD_SendData(y1 >> 8);
        LCD_SendData(y1 & 0xFF);
        LCD_SendData(y2 >> 8);
        LCD_SendData(y2 & 0xFF);
}

void LCD_Fill(uint16_t color) {
        unsigned int n, i, j;
        i = color >> 8;
        j = color & 0xFF;
        LCD_SetCursorPosition(0, 0, LCD_WIDTH - 1, LCD_HEIGHT - 1);

        LCD_SendCommand(ILI9341_GRAM);

        for (n = 0; n < LCD_PIXEL_COUNT; n++) {
                LCD_SendData(i);
                LCD_SendData(j);
        }
}


void LCD_Image(unsigned char const *img) {
      uint32_t n;

        LCD_SetCursorPosition(0, 0, LCD_WIDTH - 1, LCD_HEIGHT - 1);

        LCD_SendCommand(ILI9341_GRAM);

        for (n = 0; n < LCD_PIXEL_COUNT; n++) {
          u8 color = *img;
          img++;
                LCD_SendData(*img);
                LCD_SendData(color);
          img++;
        }
}
```

```
#define FONT_HEIGHT 25
#define FONT_WIDTH 20

uint32_t LCD_Putchar(uint32_t x, uint32_t y, char c) {
	uint32_t i, j;
	unsigned short Data;

	uint32_t offset = (c-32)*TimesNewRoman.height;
	uint16_t width = TimesNewRoman.width;

	 for (i = 0; i < TimesNewRoman.height; i++) {

	Data = TimesNewRoman.data_table[offset+i];


	for (j = 0; j < width; j++) {
	   if ((Data << j) & 0x8000) {
	      LCD_DrawPixel(x + j, (y + i), 0xFFFF);  //white
	   } else {
	      LCD_DrawPixel(x + j, (y + i), 0x0000);  //black
	   }
	}
	}

	return x+width;
}

void LCD_DrawPixel(uint16_t x, uint16_t y, uint16_t color) {
   LCD_SetCursorPosition(x, y, x, y);
   LCD_SendCommand(ILI9341_GRAM);
   LCD_SendData(color >> 8);
   LCD_SendData(color & 0xFF);
}

void LCD_DrawString(uint32_t x, uint32_t y, char *str)
{
   while(*str) {
      x = LCD_Putchar(x,y,*str++);
   }
}


//tft_lcd.h

#ifndef __TFT_LCD_H
#define __TFT_LCD_H

#include "spi1.h"
```

```
/*
spi1 miso pa6
spi1 mosi pa7
spi1 sck pa5
spi1 nss pa4
reset pa2
d/c pa3
*/

#define TFT_RESET_PIN GPIO_Pin_2
#define TFT_DC_PIN    GPIO_Pin_3
#define TFT_CS_PIN    GPIO_Pin_4

#define TFT_DC_SET    GPIO_SetBits(GPIOA, TFT_DC_PIN);
#define TFT_DC_RESET  GPIO_ResetBits(GPIOA, TFT_DC_PIN);

#define TFT_RST_SET   GPIO_SetBits(GPIOA, TFT_RESET_PIN);
#define TFT_RST_RESET GPIO_ResetBits(GPIOA, TFT_RESET_PIN);

#define TFT_CS_SET    GPIO_SetBits(GPIOA, TFT_CS_PIN);
#define TFT_CS_RESET   GPIO_ResetBits(GPIOA, TFT_CS_PIN);

#define LCD_WIDTH       240
#define LCD_HEIGHT      320
#define LCD_PIXEL_COUNT  LCD_WIDTH * LCD_HEIGHT

/// TODO: list of defines of commands
/// FIRST: test reading from display

void LCD_SendCommand(u8 com);
void LCD_SendData(u8 data);
u8 LCD_Receve();

void LCD_Init();
void LCD_InitGPIO();
void LCD_Fill(uint16_t color);
void LCD_Image(unsigned char const *img);

void LCD_DrawPixel(uint16_t x, uint16_t y, uint16_t color);
uint32_t LCD_Putchar(uint32_t x, uint32_t y, char c);
void LCD_DrawString(uint32_t x, uint32_t y,char *str);


void Delay(__IO uint32_t nTime);

//Commands
#define ILI9341_RESET                   0x01
#define ILI9341_SLEEP_OUT               0x11
```

```
#define ILI9341_GAMMA           0x26
#define ILI9341_DISPLAY_OFF     0x28
#define ILI9341_DISPLAY_ON      0x29
#define ILI9341_COLUMN_ADDR     0x2A
#define ILI9341_PAGE_ADDR       0x2B
#define ILI9341_GRAM            0x2C
#define ILI9341_MAC             0x36
#define ILI9341_PIXEL_FORMAT    0x3A
#define ILI9341_WDB             0x51
#define ILI9341_WCD             0x53
#define ILI9341_RGB_INTERFACE   0xB0
#define ILI9341_FRC             0xB1
#define ILI9341_BPC             0xB5
#define ILI9341_DFC             0xB6
#define ILI9341_POWER1          0xC0
#define ILI9341_POWER2          0xC1
#define ILI9341_VCOM1           0xC5
#define ILI9341_VCOM2           0xC7
#define ILI9341_POWERA          0xCB
#define ILI9341_POWERB          0xCF
#define ILI9341_PGAMMA          0xE0
#define ILI9341_NGAMMA          0xE1
#define ILI9341_DTCA            0xE8
#define ILI9341_DTCB            0xEA
#define ILI9341_POWER_SEQ       0xED
#define ILI9341_3GAMMA_EN       0xF2
#define ILI9341_INTERFACE       0xF6
#define ILI9341_PRC             0xF7


#endif /* __TFT_LCD_H */
```

Лабораторна робота № 3

ДОСЛІДЖЕННЯ ВУЗЛА ВВЕДЕННЯ ВІДЕОІНФОРМАЦІЇ В МПС

МЕТА РОБОТИ: ознайомлення з технічними засобами введення відеозображень в мікропроцесорні системи (МПС), архітектурою та особливістю функціонування цифрової відеокамери типу OV7670, інтерфейсом підключення цифрової відеокамери до МПС, з функціонуванням відеокамери в режимі введення відеозображень та виведенням на кольоровий дисплей в реальному часі. Ознайомитися з основними етапами та отримати навики розроблення драйвера вузла введення/виведення відеозображення в МПС.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з лабораторним стендом на основі ***на відлагоджувального модуля*** **STM32F429 Discovery.**
2. Виконати у вказаному порядку основні етапи підготовки базової програми для введення/виведення відеозображення в МПС для з використанням STM32CubeMX (**ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ**).
3. Запрограмувати з допомогою утиліти **STM32 ST-LINK Utility** тестову програму **ov7670-il9341.hex** для введення відеозображень з цифрової відеокамери **OV7670** та виводу на кольоровий графічний індикатор **ILI9341**.
4. Використовуючи програмні модулі керування цифровою відеокамерою **OV7670** та графічним індикатором **ILI9341** згідно до завдання керівника підготувати програму для введення відеозображень з цифрової відеокамери **OV7670** та виводу на кольоровий графічний індикатор **ILI9341** ***відлагоджувального модуля*** **STM32F429 Discovery**.
5. Відповісти на контрольні запитання.
6. Захистити лабораторну роботу.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Ознайомитися з методичними матеріалами до лабораторної роботи
2. Скачати STM32CubeMX з сайту

https://my.st.com/cas/login?service=https%3A%2F%2Fmy.st.com%2Fcontent%2Fmy_st_com%2Fen%2Fproducts%2Fdevelopment-tools%2Fsoftware-development-tools%2Fstm32-software-development-tools%2Fstm32-configurators-and-code-generators%2Fstm32cubemx.html

3. Для цього необхідно відкрити на вказаному сайті власний Account (**Create Account**)
4. Заінсталювати STM32CubeMX на власному ПК.
5. Ознайомитися з пакетом STM32CubeMX

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Основні компоненти лабораторного стенда.
2. Основні параметри цифрової відеокамери **OV7670.**
3. Схема підключення цифрової відеокамери **OV7670** до відлагоджувального модуля **STM32F429 Discovery** лабораторного стенда**.**

4. Основні етапи підготовки програми в STM32CubeMX
5. Назвати засоби програмування пам’яті на кристалі мікропроцесорного компонента
6. Пояснити роботу підготовленої та відлагодженої програми.

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

1. Запустити **STM32CubeMX:**

*Рис. 3.1. Стартове вікно **STM32CubeMX***

2. **Поетапно** виконати кроки по підготовці для IDE Keil 4 базового програмного коду для введення відеозображень з цифрової відеокамери **OV7670** та виводу на кольоровий графічний індикатор **ILI9341** відлагоджувального модуля **STM32F429 Discovery** :

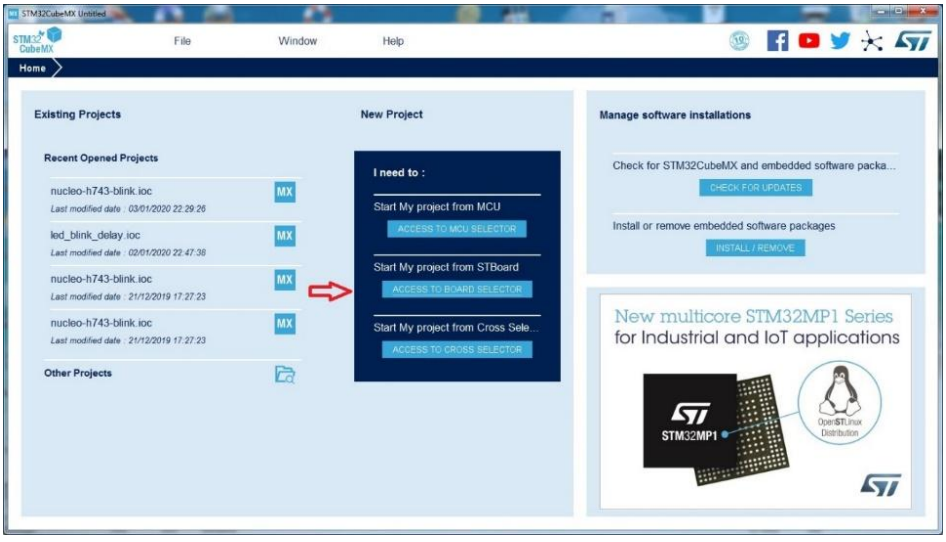


Рис. 3.2. Вибір апаратного відлагоджувального модуля

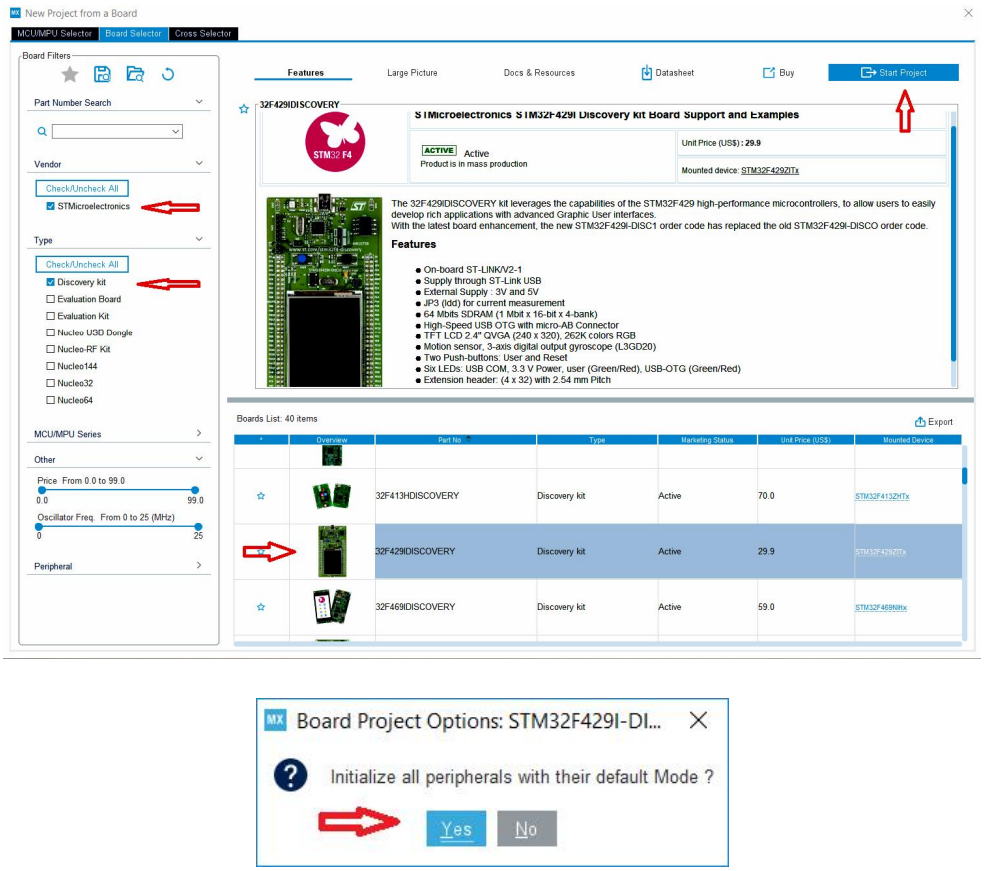


Рис. 3.3. вибір апаратного відлагоджувального модуля ***STM32F429 Discovery***

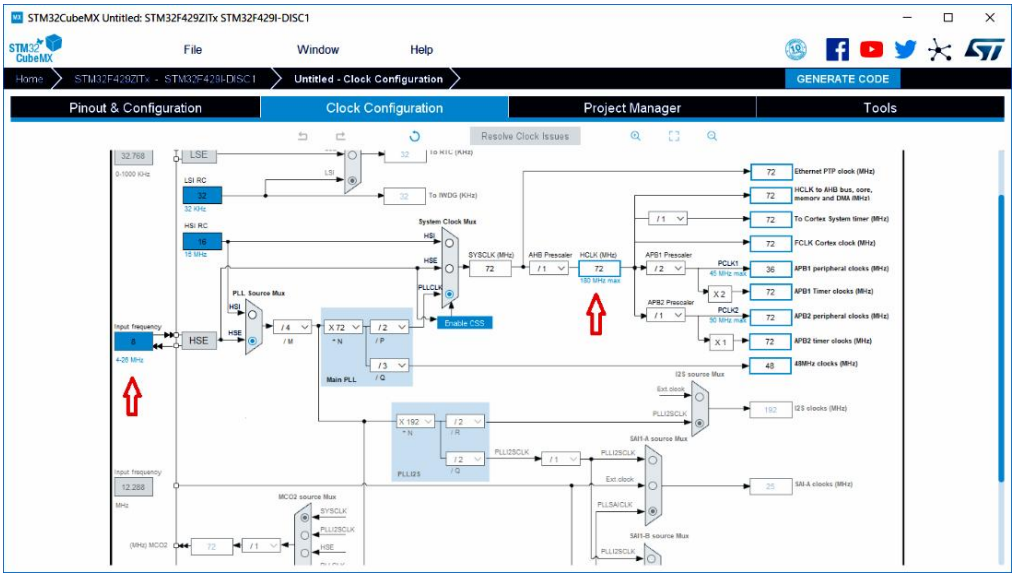


Рис. 3.4. Вибір параметрів системи синхронізації

– вибір інтерфесу I2C та ліній I/O для керування і введення відеозображень з цифрової відеокамери **OV7670** та виводу на кольоровий графічний індикатор **ILI9341** відлагоджувального модуля **STM32F429 Discovery** та при необхідності ліній інших вузлів мікропроцесорної системи відповідного функціонального призначення

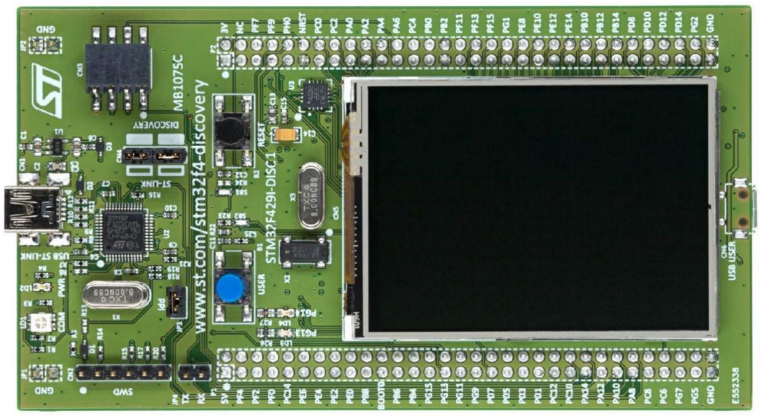


*Рис. 3.5. Модуль **STM32F429 Discovery***

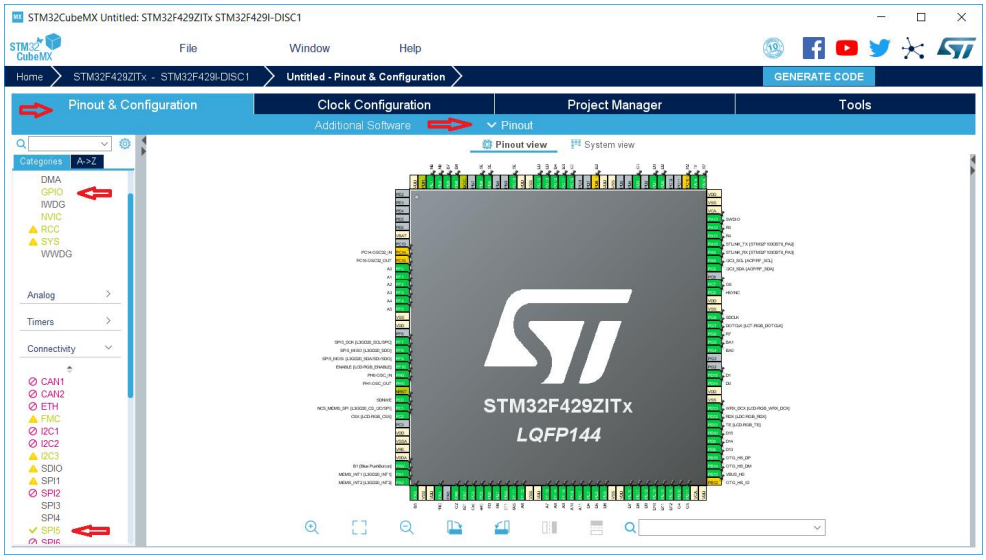


Рис. 3.6. Вибір інтерфесу I2C та ліній I/O

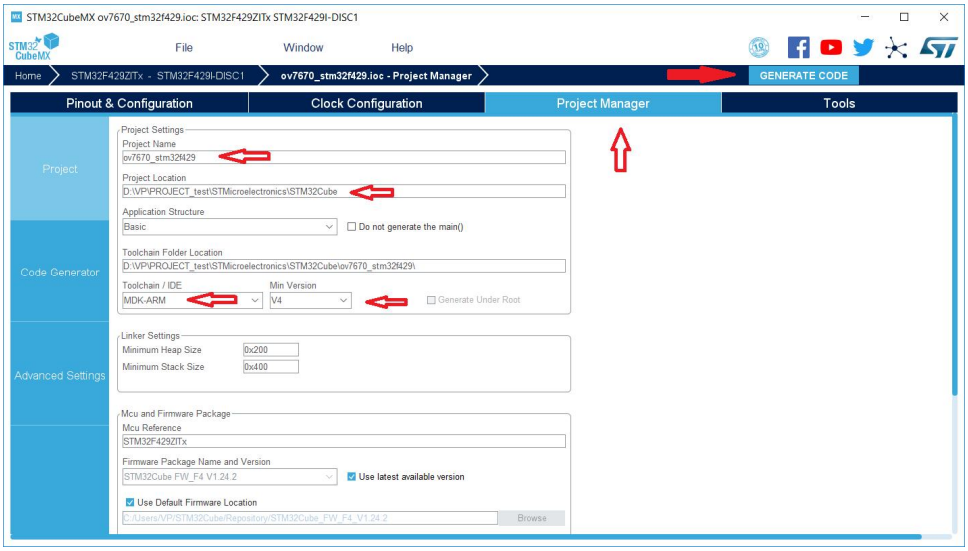


Рис. 3.7. Вибір назви проєкту та середовища IDE для генерування коду

Рис. 3.8. Процес генерування коду

Рис. 3.9. Запуск проєкту

Рис. 3.10. Розміщення згенерованого IDE проєкту в папці проєкту STM32CubeMX

Рис. 3.11. Файл запуску IDE проєкту

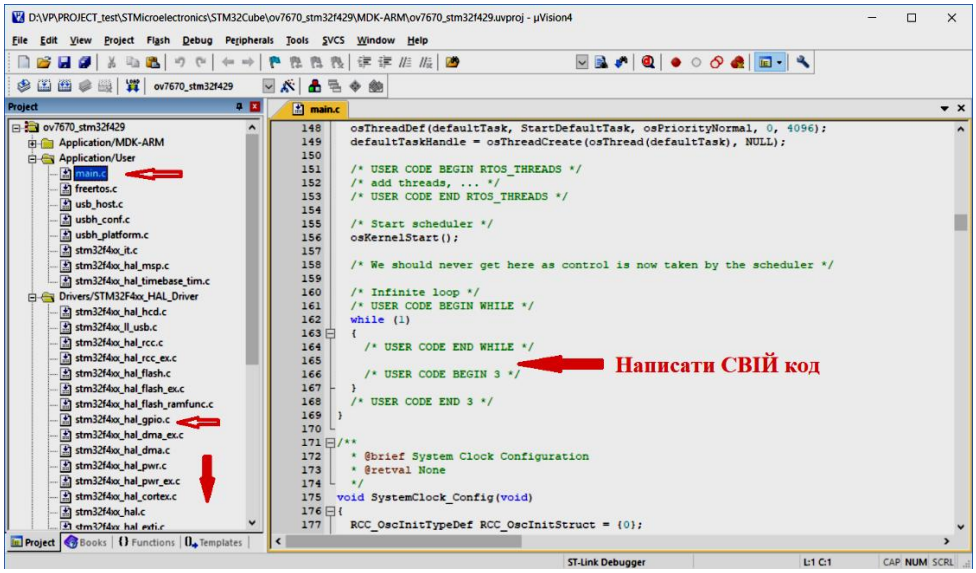


*Рис. 3.12. Вікно модуля **main.c***

– запуск компіляції, виправлення помилок при їх наявності, перевірка та при необхідності корекція відповідних модулів ініціалізації SPI, I/O та інших вузлів у відповідності до мікропроцесорної системи конкретного функціонального призначення

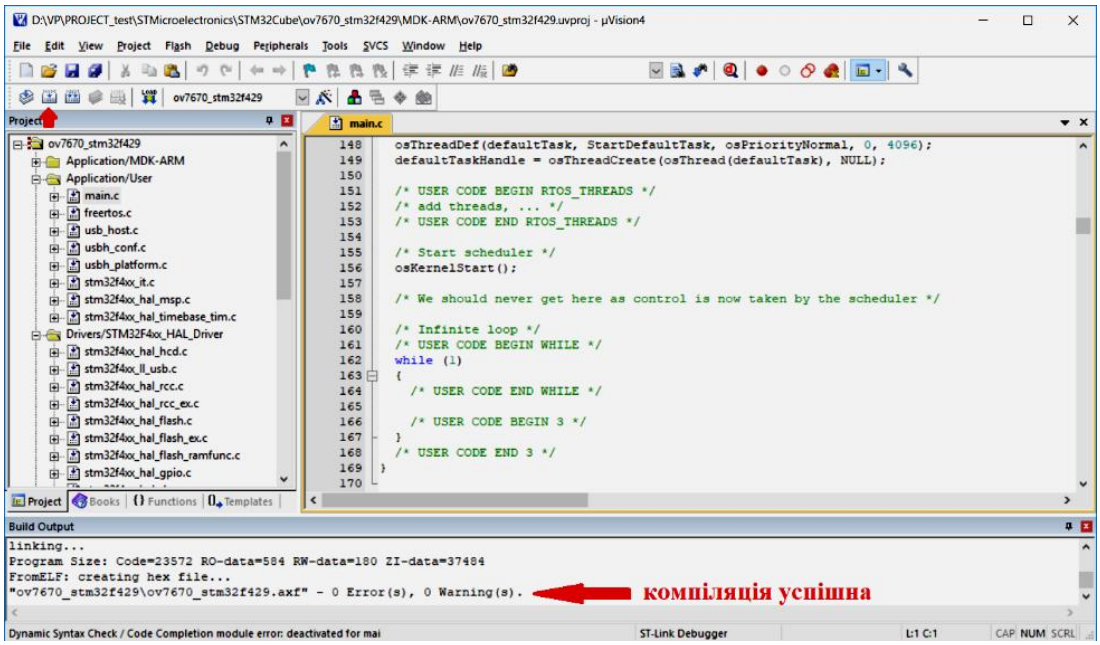


Рис. 3.13. Запуск компіляції

Модулі ініціалізації основних вузлів

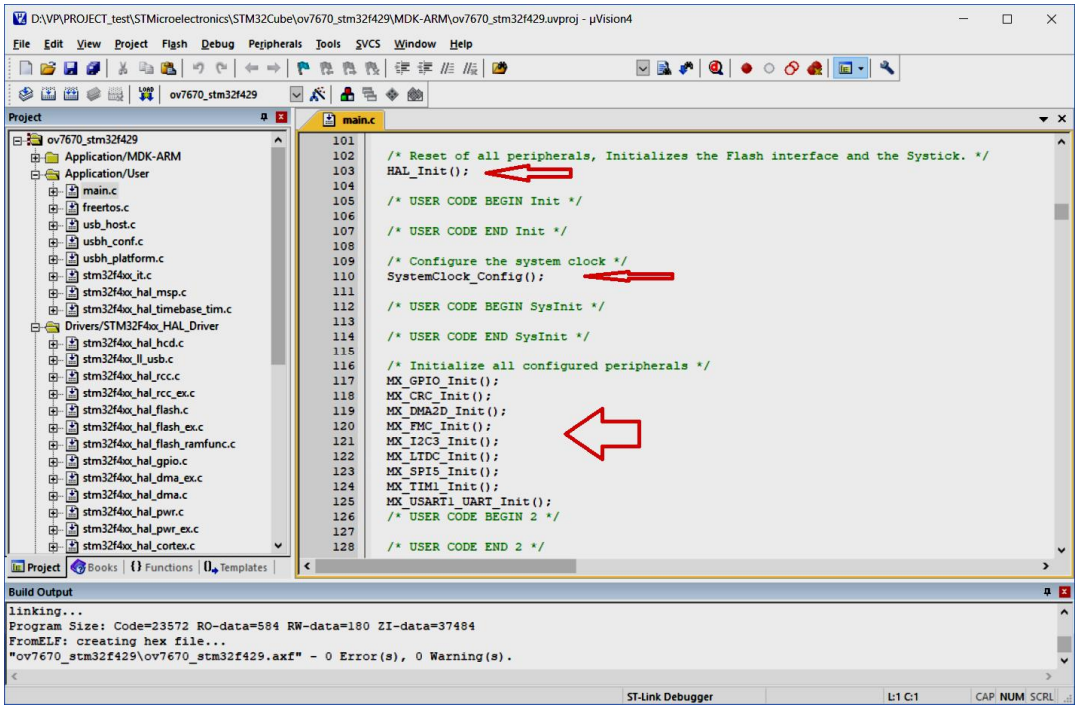


Рис. 3.14. Модулі ініціалізації основних вузлів

Підключити стенд до гнізда USB на ПК та запустити відлагоджувач (вікно **main.c** після вдалого запуску)

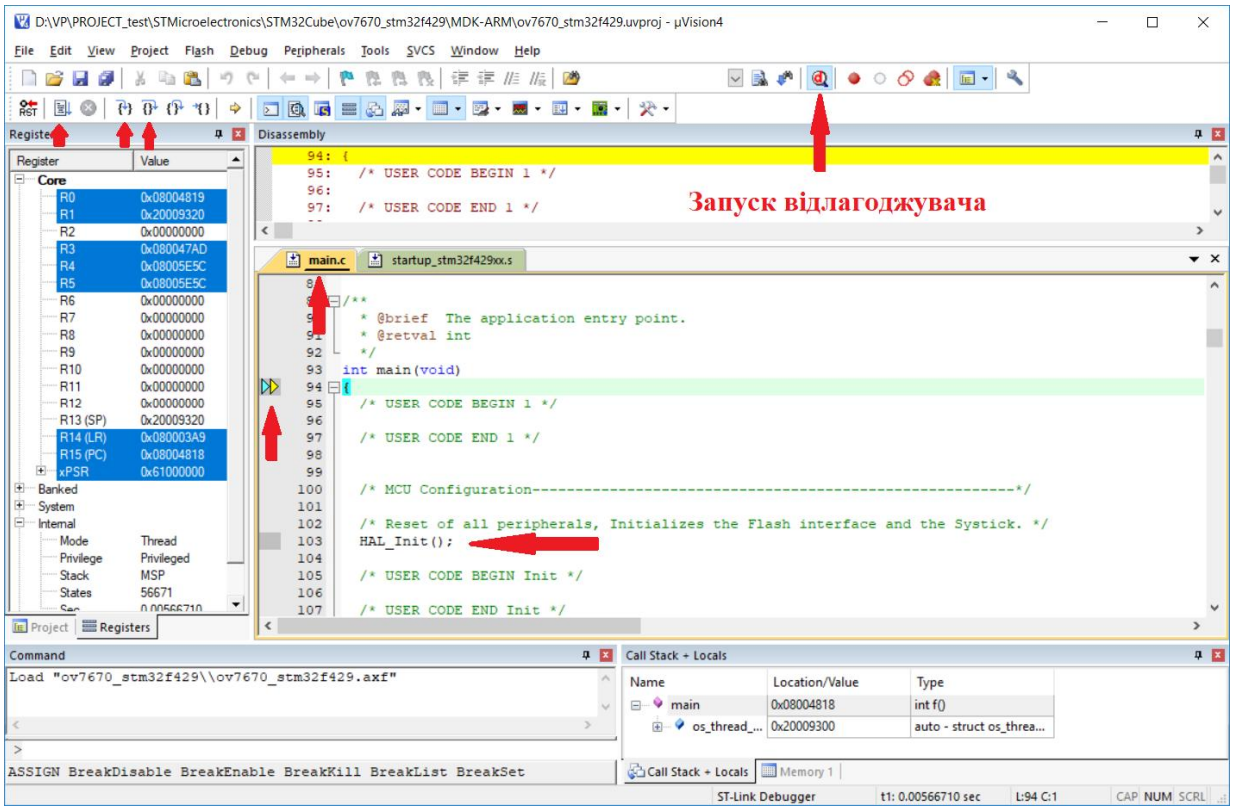


Рис. 3.15. Запуск відлагоджувача

3. **Запис завантажувального модуля програми в форматі *.hex в пам'ять на кристалі мікропроцесорного компонента**

Програмування пам'яті на кристалі мікропроцесорного компонента можна здійснювати різними засобами:

– в процесі запуску відлагоджувача в режимі роботи з апаратурою;

– інтегрованим програматором в середовище IDE;

– окремою програмою, призначеною для роботи з відповідним мікропроцесорним сімейством, наприклад, утилітою **STM32 ST-LINK Utility.exe**:

Вікно запуску **STM32 ST-LINK Utility**

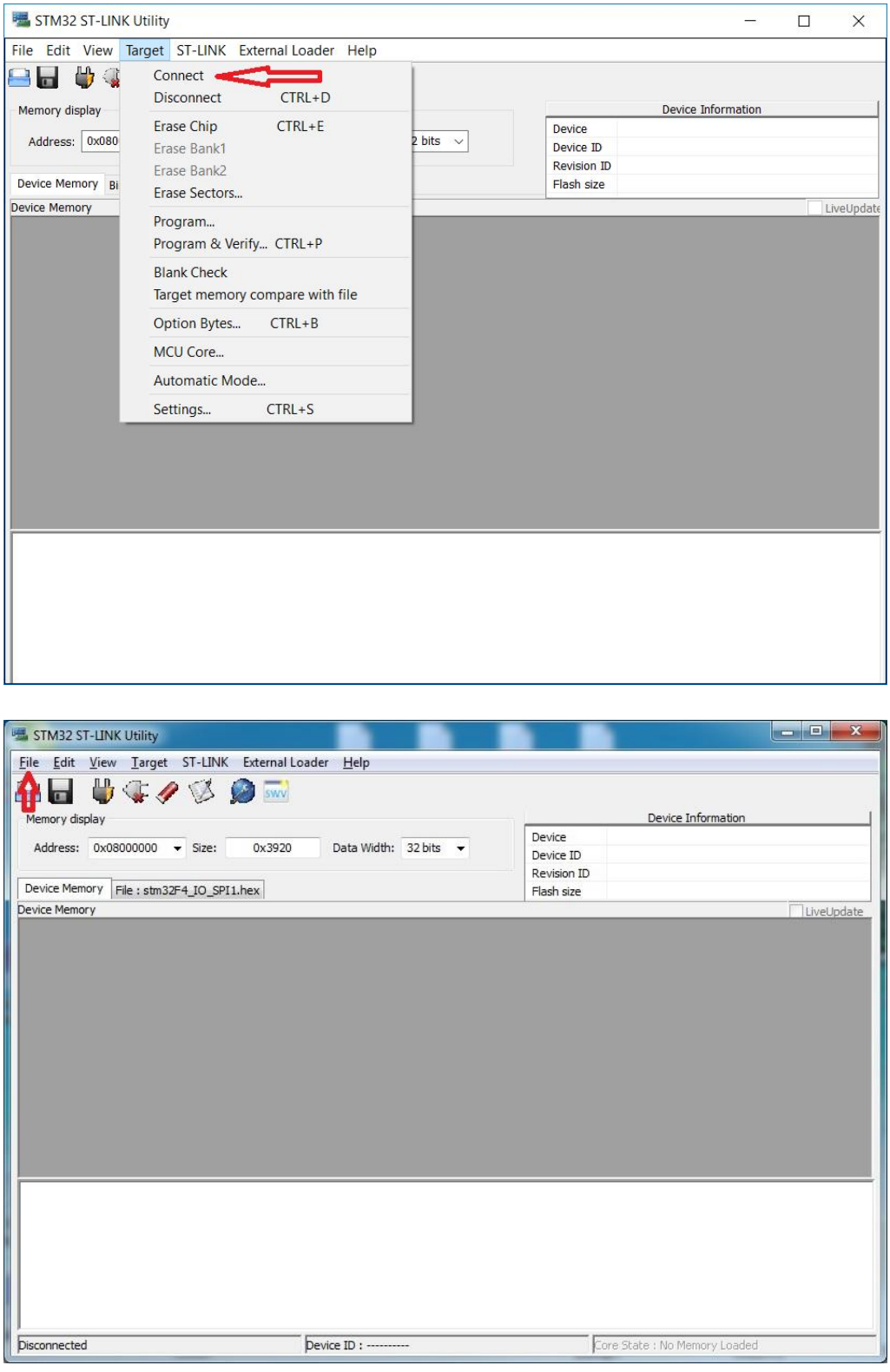


*Рис. 3.16. Запуск **STM32 ST-LINK Utility***

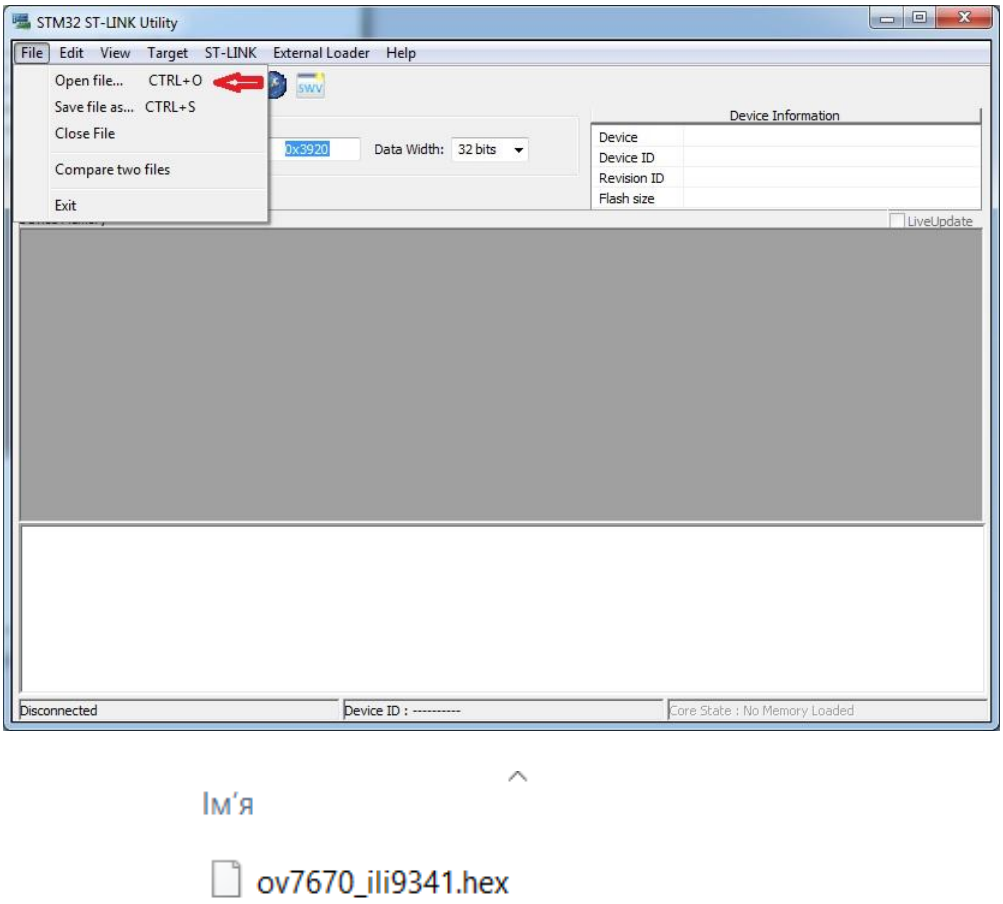


*Рис. 3.17. Вибір файла **в форматі *.hex** для програмування в пам'ять кристалу*

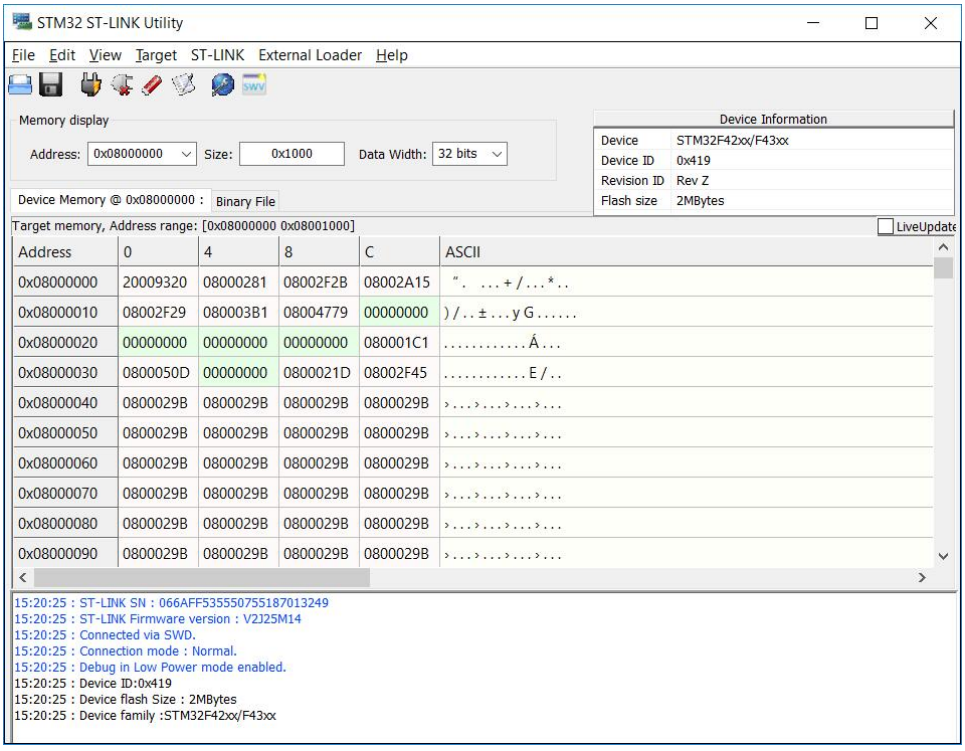


Рис. 3.18. Вибір етапу програмування та верифікації

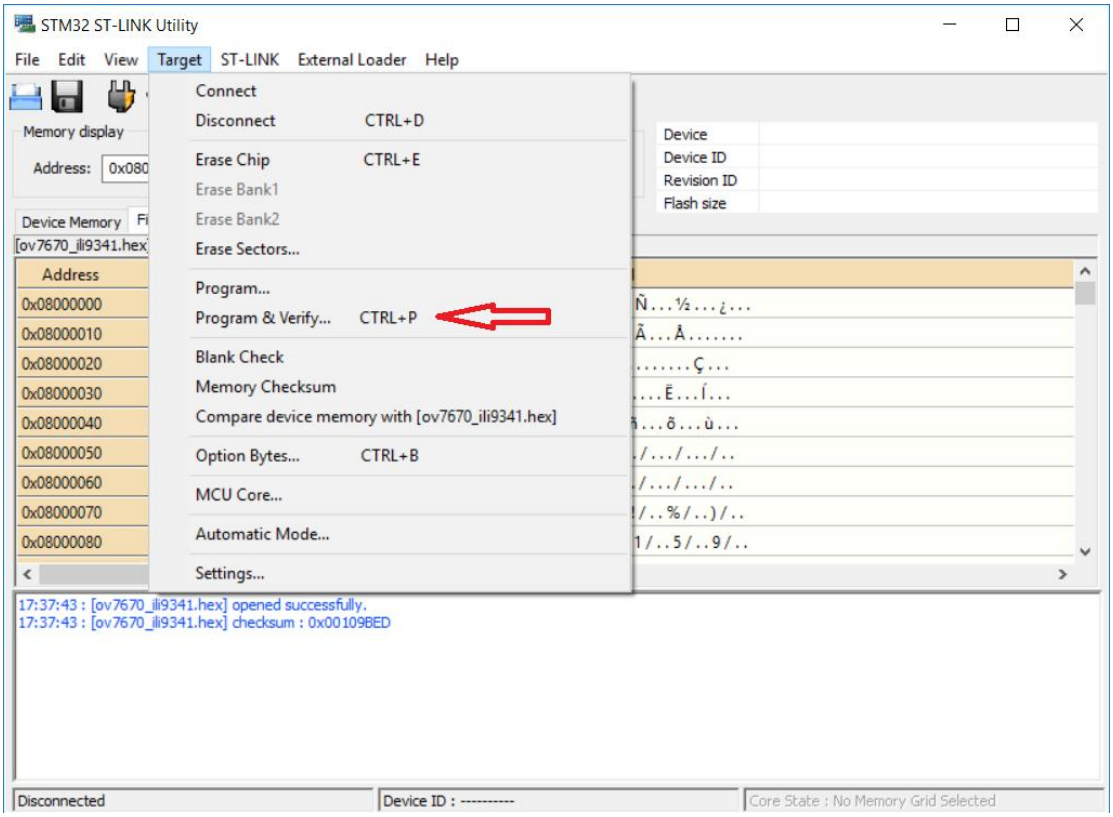


Рис. 3.18 (продовження). Вибір етапу програмування та верифікації

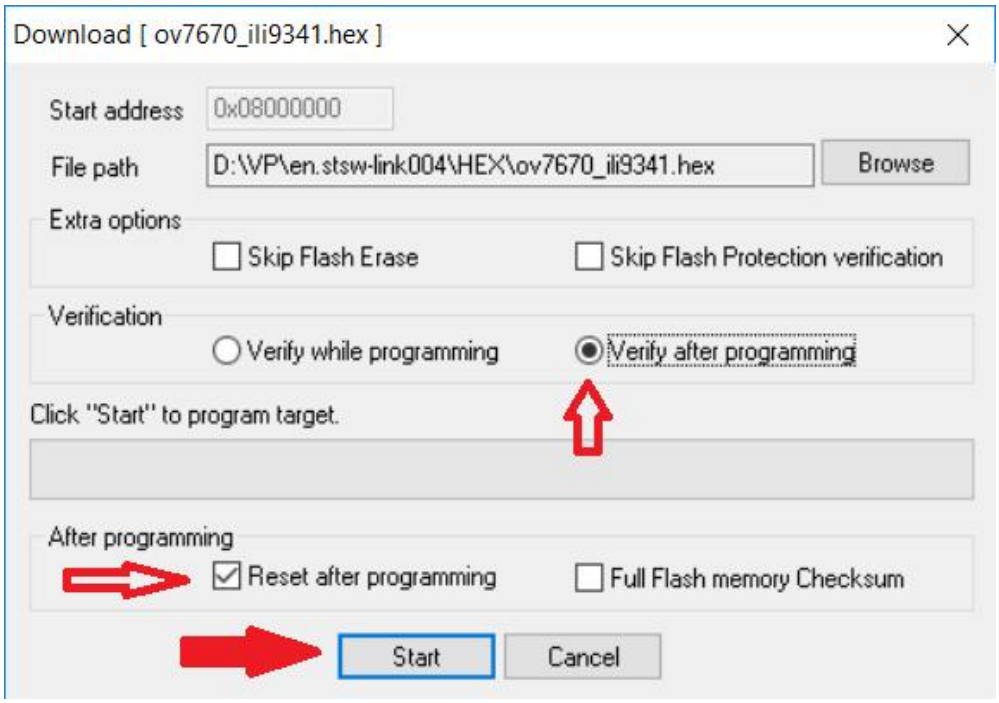


Рис. 3.19. Вибір режимів програмування
*та запуск процесу програмування "**Start**"*

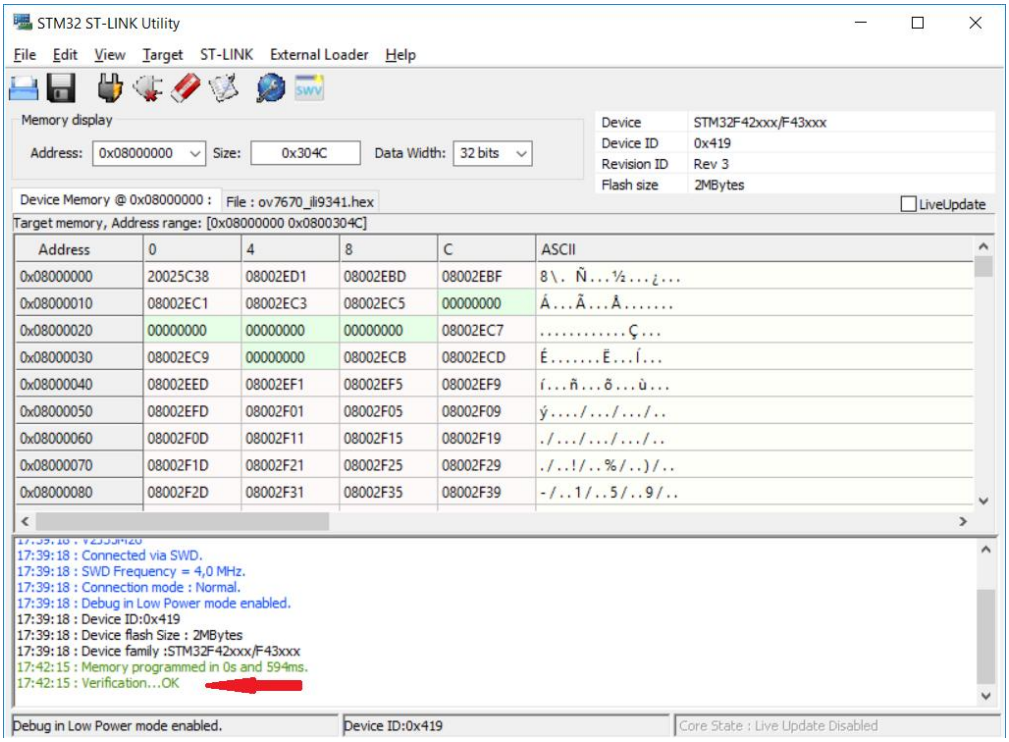


Рис. 3.20. Успішне завершення процесу програмування та верифікації запрограмованого коду з вхідним ****.hex***

МЕТОДИЧНІ МАТЕРІАЛИ

Введення відеоінформації в мікропроцесорну систему (МПС) можна реалізувати на основі аналогової чи цифрової відеокамери. Існують різні типи відеокамер з різними технічними характеристиками та експлуатаційними параметрами.

Аналогові відеокамери

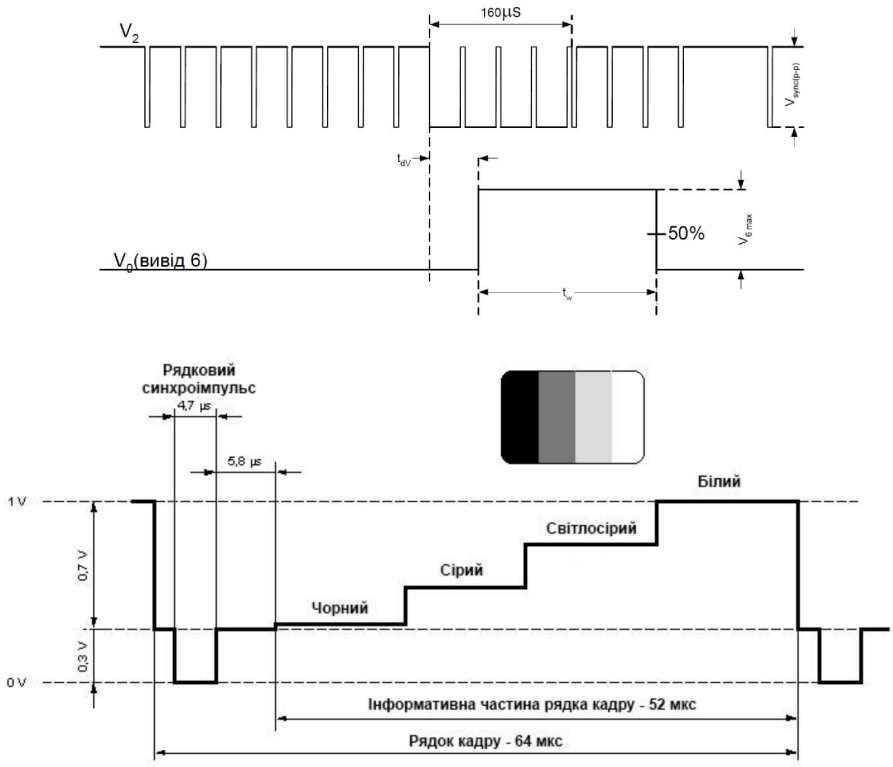
Прикладом застосування аналогової відеокамери є відеокамери огляду деяких автомобілів тощо.

Аналогова відеокамера формує аналоговий відеосигнал, який для введення в МПС необхідно перетворити в цифрову форму з використанням швидкодіючого аналого-цифрового перетворювача (АЦП). Відеосигнал монохромної (чорно-білої) аналогової відеокамери містить інформацію про яскравість елементів зображення, а також синхросигнали рядкових, кадрових та керуючих синхроімпульсів. Повний кольоровий телевізійний сигнал крім відеосигналу містить піднесучу, промодульовану сигналом кольору, що містить інформацію про колір елементів зображення, а також сигнал синхронізації кольору.

Цифрові відеокамери

Цифрові відеокамери формують на виході віеозабраження в цифровому коді в різноманітних форматах. Передача відеоінформації в МПС здійснюється через паралельний чи швидкодіючий послідовний інтерфейс, наприклад, SPI. Для керування режимами функціонування цифрової відеокамери часто використовується інтерфейс I2C.

В лабораторній роботі досліджується введення відеоінформації в МПС з допомогою цифрової відеокамери OV7670, див. відповідний підрозділ методичних матеріалів.



3.21. Відеосигнал монохромної відеокамери

Лабораторний стенд

Стенд реалізовано на базі відлагоджувального модуля **STM32F429DISCOVERY** з підключеною цифровою відеокамерою **OV7670** та кольоровиим графічним індикатором **ILI9341**. Підключення стенду здійснюється **USB** кабелем до любого порту **USB 2.0** чи **USB 3.0** комп'ютера.

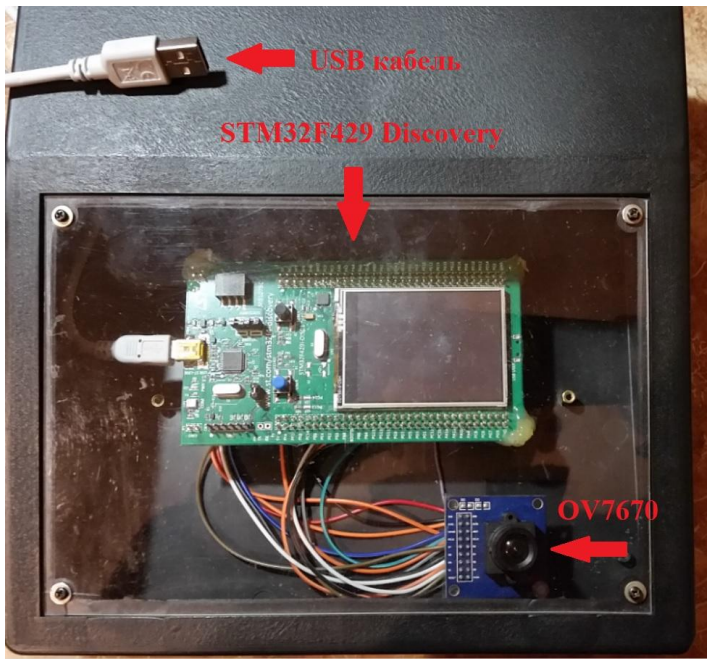


Рис. 3.22. Лабораторний стенд

Відлагоджувальний модуль STM32F429DISCOVERY

Відлагоджувальний модуль **STM32F429DISCOVERY** призначений для відлагодження в реальному часі програм для мікропроцесорних систем (МПС) різного функціонального призначення з метою розпаралелювання процесу розроблення апаратної частини МПС та програмного забезпечення до неї. **STM32F429DISCOVERY** реалізовано на базі мікроконтролера з ядром **Cortex-M4F** типу **STM32F429ZIT6** в корпусі **LQFP144**. Структурна схема та зовнішній вигляд **STM32F429DISCOVERY** зображені на рис.

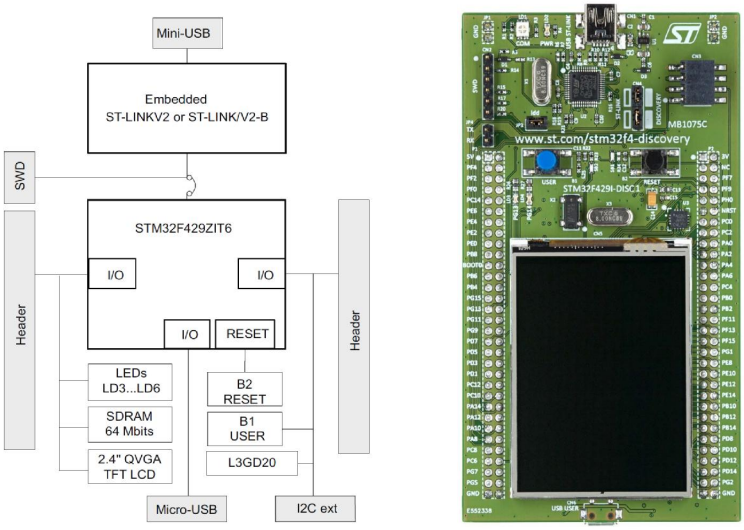
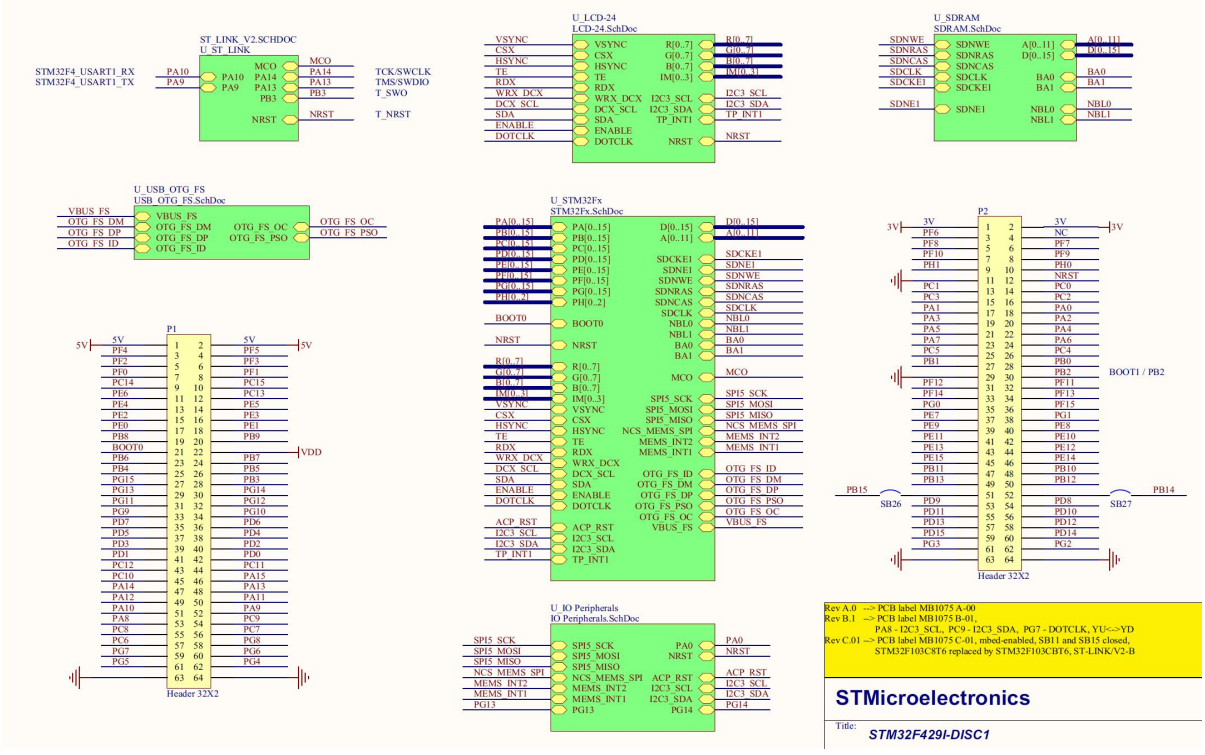
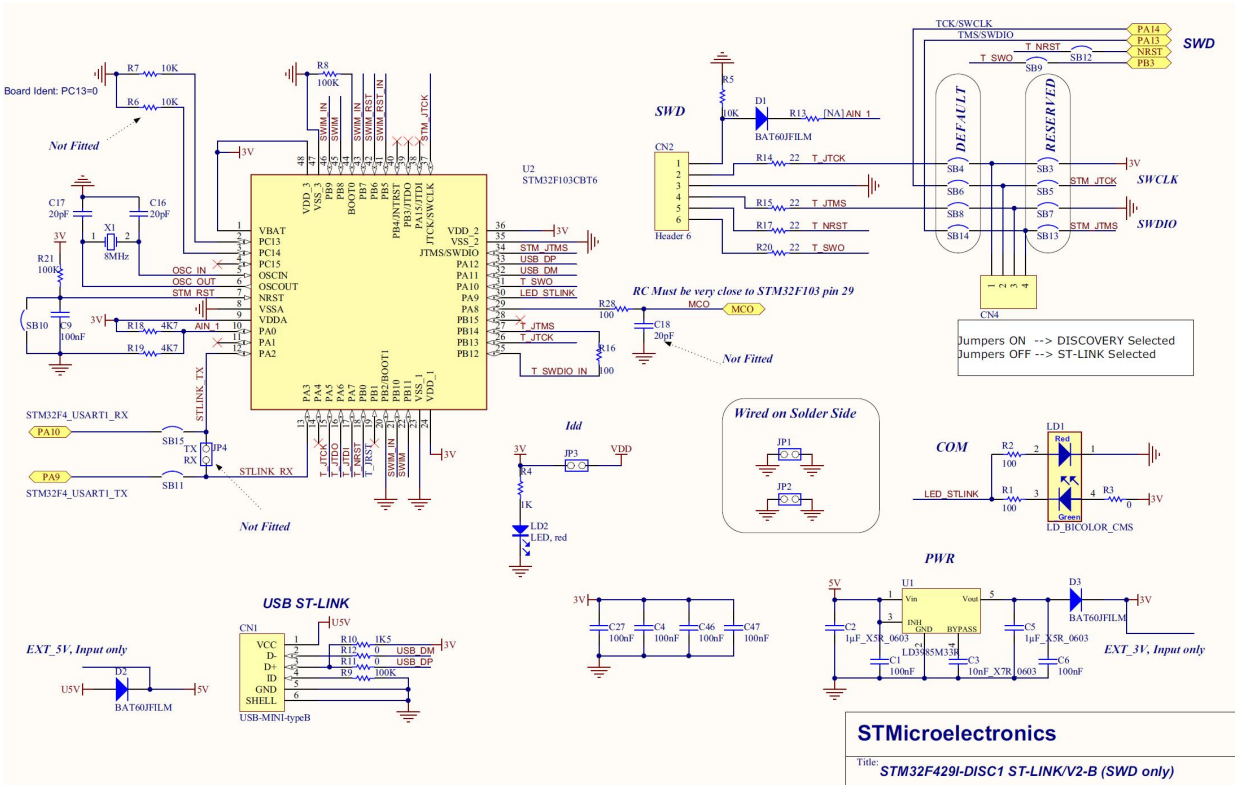


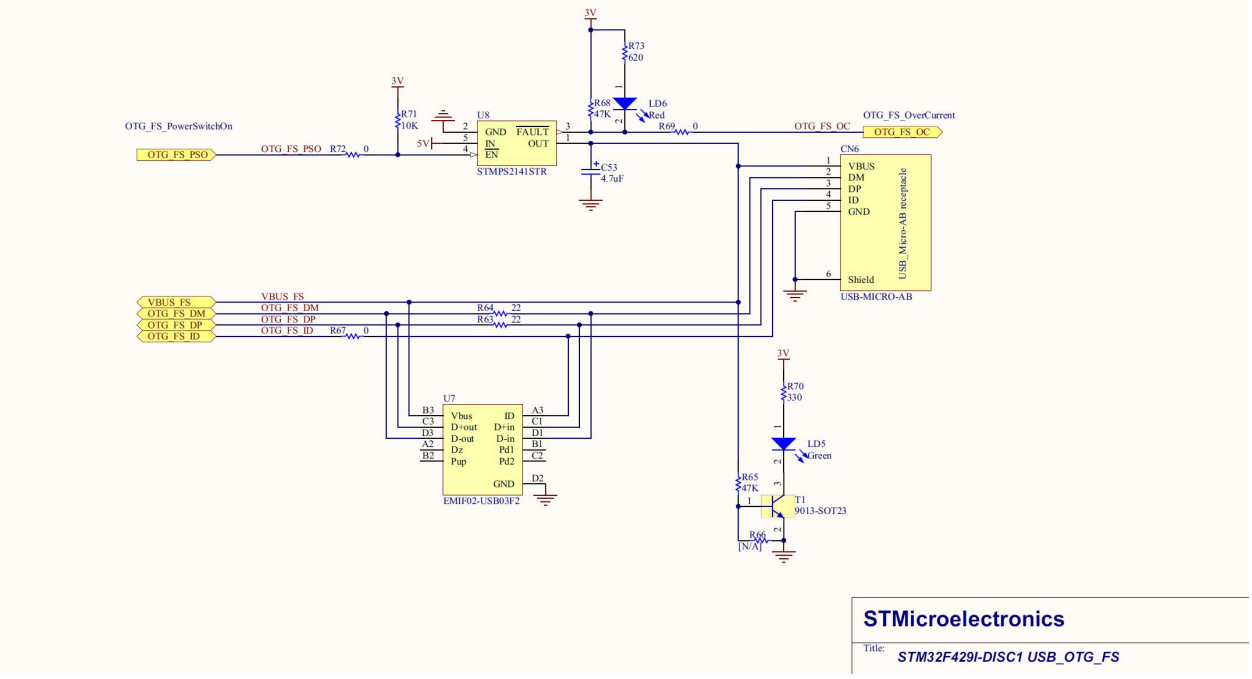
*Рис. 3.23. Структурна схема та зовнішній вигляд **STM32F429DISCOVERY***

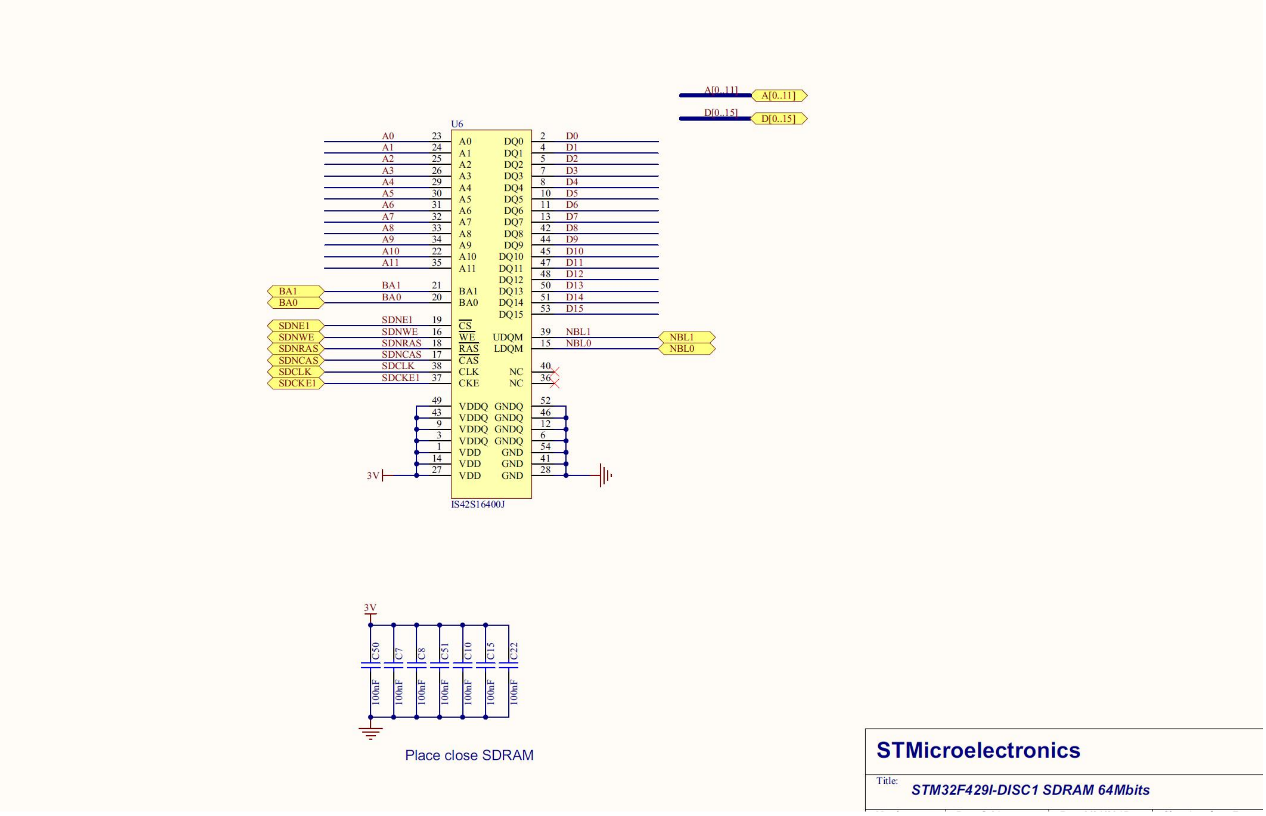
Нижче показано схеми відлагоджувального модуля **STM32F429DISCOVERY** [3.2].

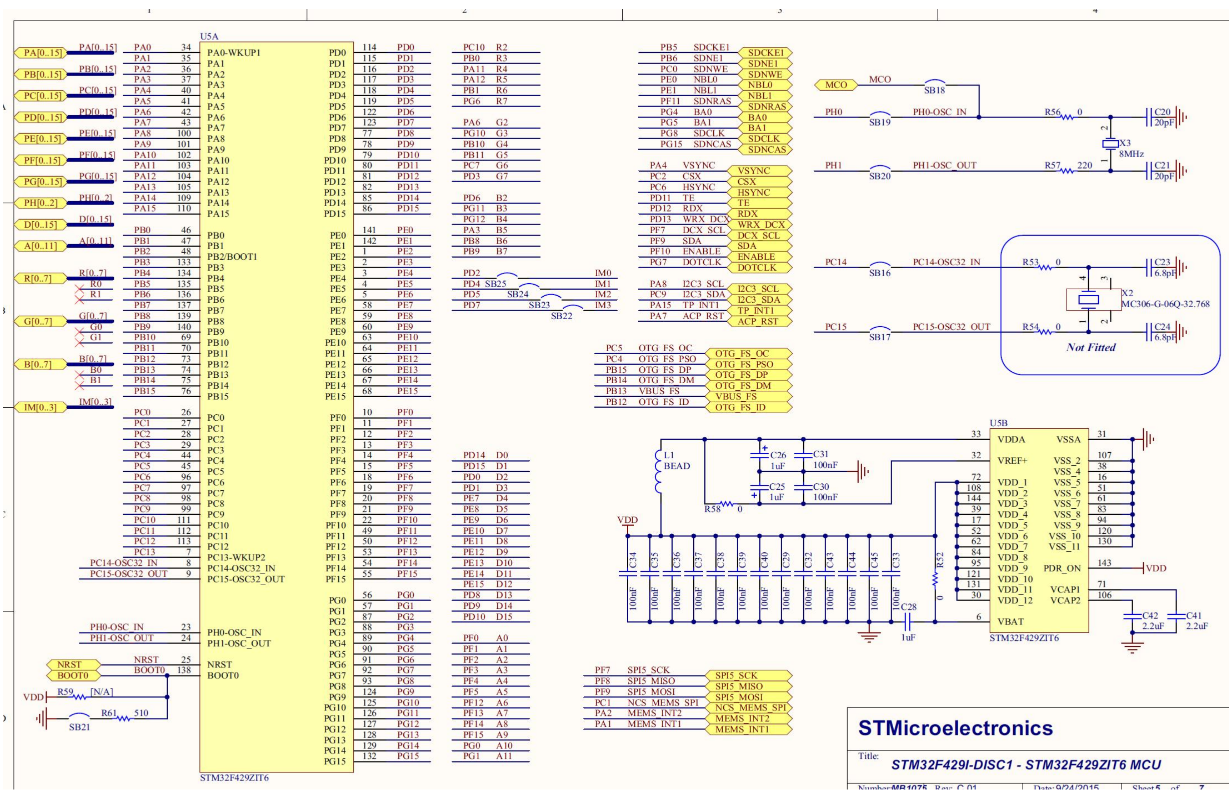
Схеми **STM32F429DISCOVERY**

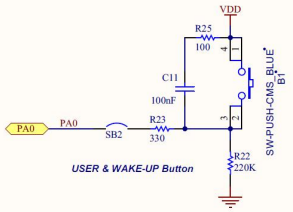


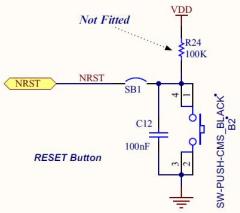

Board Ident: PC13=0
Not Fitted
SWD
DEFAULT
RESERVED
SWCLK
SWDIO
U2
STM32F103CBT6
Header 6
CN4
Jumpers ON --> DISCOVERY Selected
Jumpers OFF --> ST-LINK Selected
RC Must be very close to STM32F103 pin 29
Not Fitted
Idd
Wired on Solder Side
COM
LD_BICOLOR_CMS
PWR
EXT_5V, Input only
EXT_3V, Input only
USB ST-LINK
USB-MINI-typeB
BAT60JFILM
STMicroelectronics
Title:
STM32F429I-DISC1 ST-LINK/V2-B (SWD only)

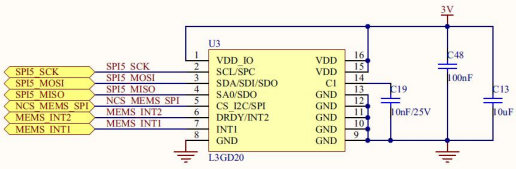

OTG_FS_PowerSwitchOn
OTG_FS_PSO
OTG_FS_OverCurrent
OTG_FS_OC
STMPS2141STR
USB_Micro-AB receptacle
USB-MICRO-AB
VBUS_FS
OTG_FS_DM
OTG_FS_DP
OTG_FS_ID
EMIF02-USB03F2
LD5
Green
T1
9013-SOT23
STMicroelectronics
Title:
STM32F429I-DISC1 USB_OTG_FS


U6
IS42S16400J
Place close SDRAM
STMicroelectronics
Title: STM32F429I-DISC1 SDRAM 64Mbits

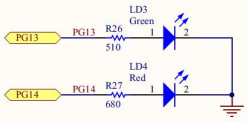

U5A
STM32F429ZIT6
U5B
STM32F429ZIT6
Not Fitted
STMicroelectronics
Title: STM32F429I-DISC1 - STM32F429ZIT6 MCU





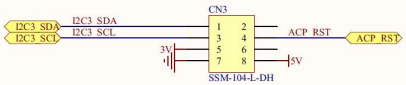


MEMS



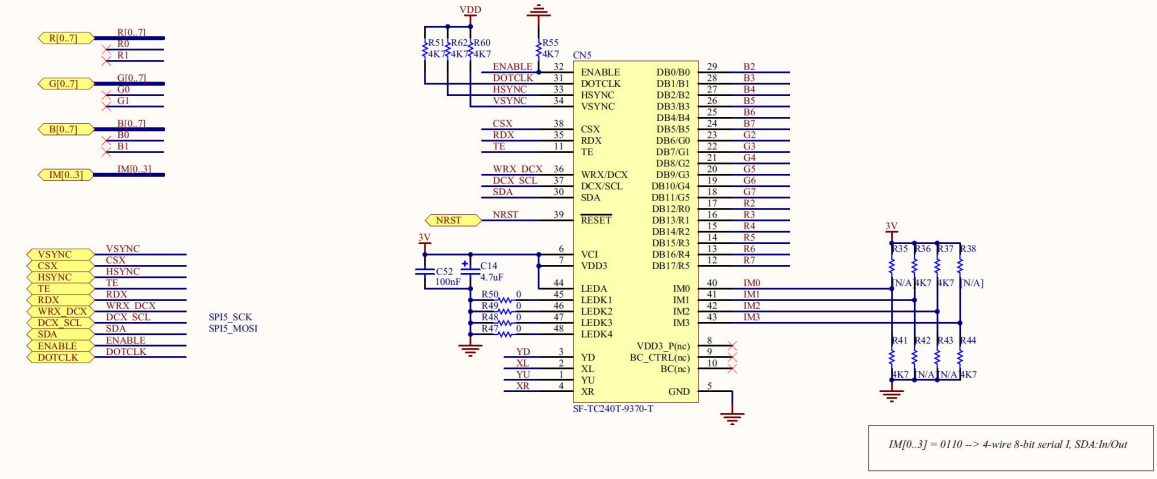
LEDs

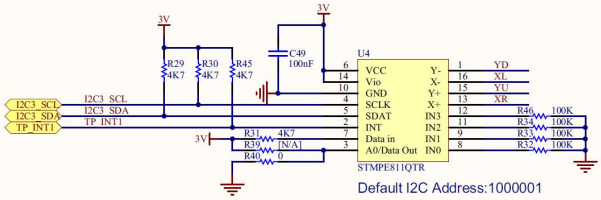
ACP/RF E2P Connector



STMicroelectronics

Title: ***STM32F429I-DISC1 Peripherals***





STMicroelectronics

Title: ***STM32F429I-DISC1 LCD 2.4"***

Цифрова відеокамера OV7670

Таблиця 3.1. Основні параметри OV7670

Роздільна здатність	640x480 (VGA)
Формати виводу	YUV/YCbCr 4:2:2 RGD565/555 GRB 4:2:2 Raw RGB Data
Максимальна швидкість передачі	30 fps (VGA)
Об’єктив	1/6”
Кут зору	24 градуси
Живлення	3.3 V

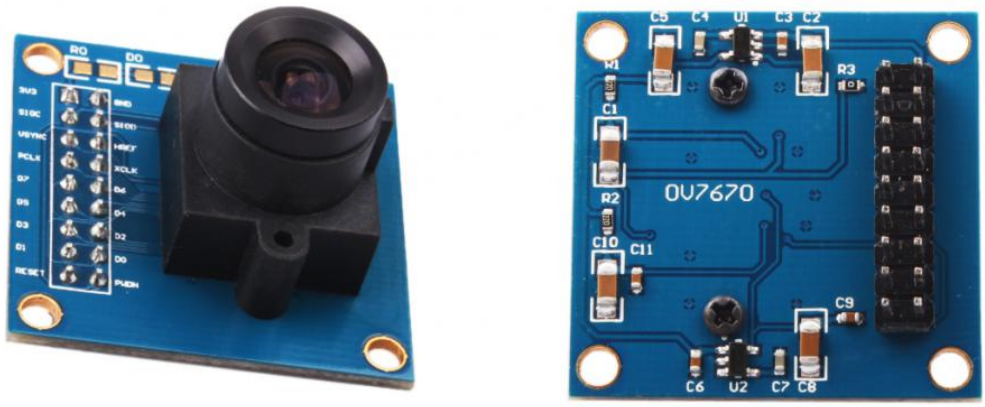


Рис. 3.24. Зовнішній вигляд цифрової відеокамери OV7670

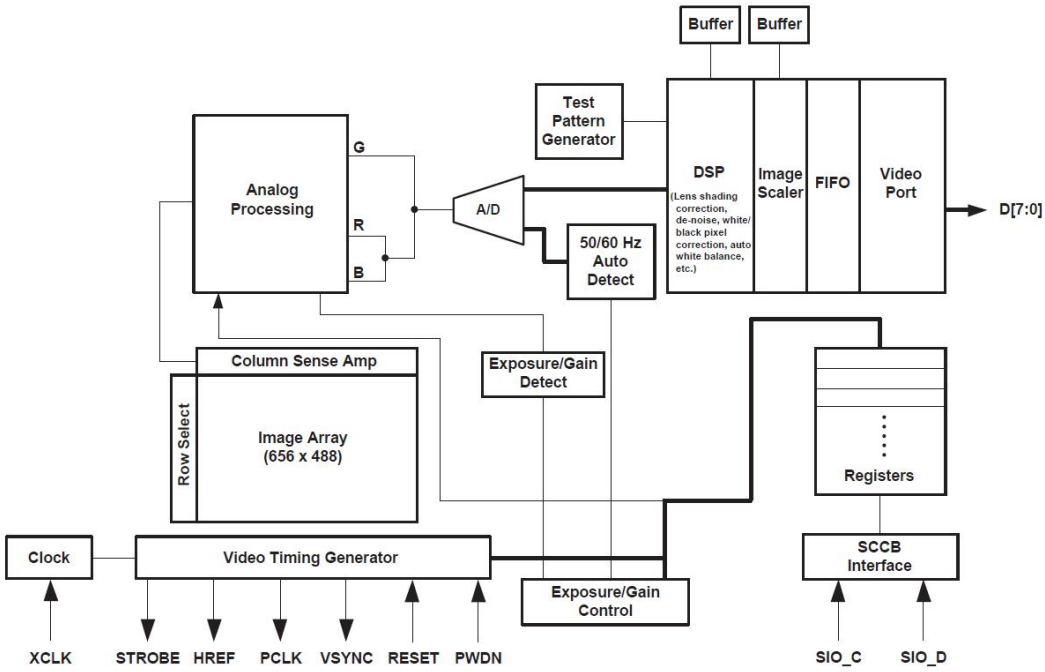


Рис. 3.25. Структурна схема цифрової відеокамери OV7670

Таблиця 3.2. Контакти підключення відеокамери OV7670

1	3.3 V	**2**	GND	**3**	SCL	**4**	SDA
5	VSYNC	**6**	HREF	**7**	PCLK	**8**	XCLK
9	D7	**10**	D6	**11**	D5	**12**	D4
13	D3	**14**	D2	**15**	D1	**16**	D0
17	RESET	**18**	PWON				

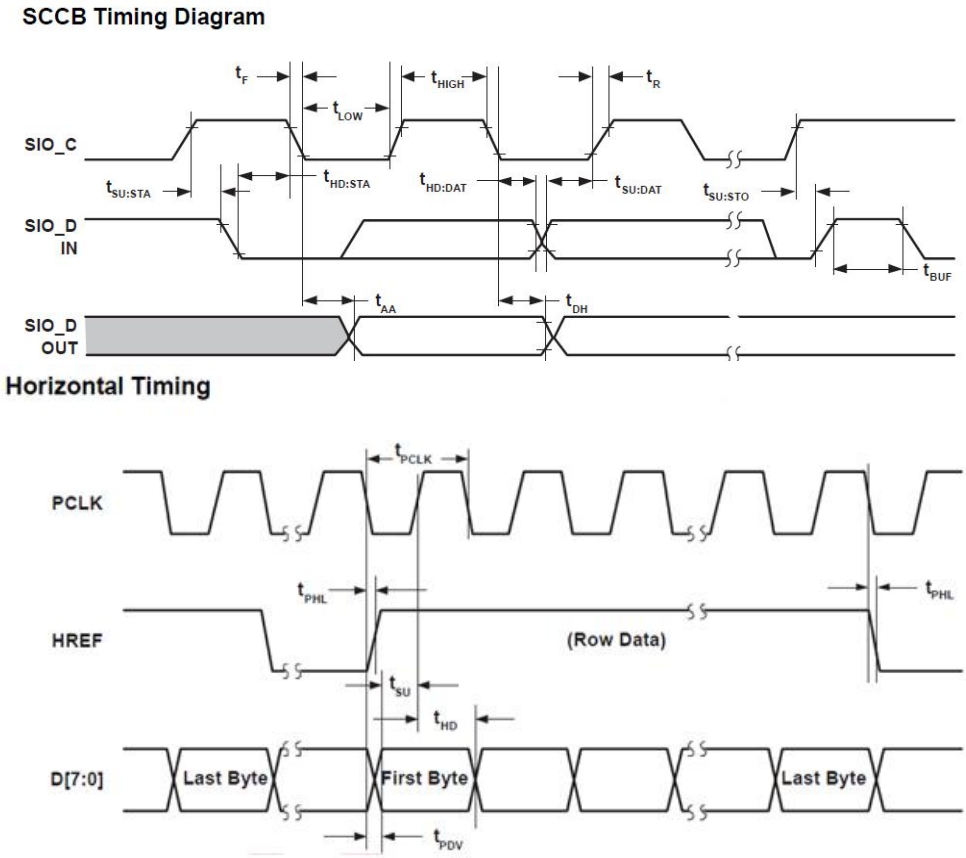


Рис. 3.26а. Часові діаграми відеокамери OV7670

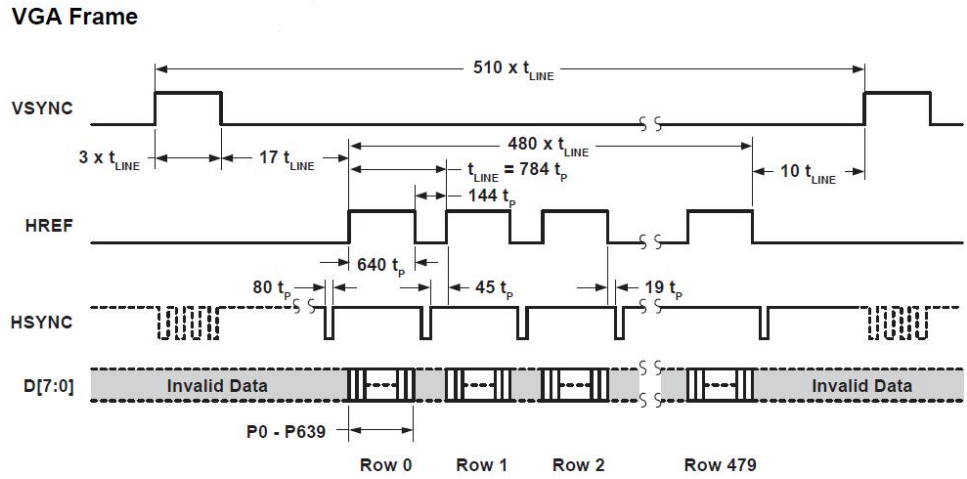


Рис. 3.26б. Часові діаграми відеокамери OV7670

Таблиця 3.3. Підключення цифрової відеокамери OV7670
до контактів STM32F429DISCOVERY

PB7	VSYNC	*PB6*	D5
PA4	HREF	*PE4*	D4
PA6	PCLK	*PC9*	D3
PA8	XCLK	*PC8*	D2
PE6	D7	*PC7*	D1
PE5	D6	*PC6*	D0

PB8	SCL (SIOC)
PB9	SDA (SIOD)

Література

3.1. STM32F429ZI.pdf
3.2. STM32F429I-DISC1.pdf
3.3. OV7670.pdf
3.4. OV7670-programming.pdf
3.5.Конспект лекцій з дисципліни “Мікропроцесорні системи”

Лістинги програмних модулів керування цифровою відеокамерою OV7670 та кольоровиим графічним індикатором ILI9341

Програмні модулі для керування цифровою відеокамерою OV7670

```
/*
*  CAMERA WIRING:
*  3V3      -  3V     ;   GND    -  GND
*  SIOC     -  PB8    ;   SIOD   -  PB9
*  VSYNC    -  PB7    ;   HREF   -  PA4
*  PCLK     -  PA6    ;   XCLK   -  PA8
*  D7       -  PE6    ;   D6     -  PE5
*  D5       -  PB6    ;   D4     -  PE4
*  D3       -  PC9    ;   D2     -  PC8
*  D1       -  PC7    ;   D0     -  PC6
*  RESET    -  -----  ;   PWDN   -  -----
*
*     ======================================================================
*/

#INCLUDE "OV7670_CONTROL.H"

// IMAGE BUFFER
VOLATILE UINT16_T FRAME_BUFFER[IMG_ROWS*IMG_COLUMNS];

CONST UINT8_T OV7670_REG [OV7670_REG_NUM][2] = {
        {0X12, 0X80},        //RESET REGISTERS

        // IMAGE FORMAT
        {0X12, 0X14},        //QVGA SIZE, RGB MODE
        {0X40, 0XD0},        //RGB565
        {0XB0, 0X84},        //COLOR MODE

        // HARDWARE WINDOW
        {0X11, 0X01},        //PCLK SETTINGS, 15FPS
        {0X32, 0X80},        //HREF
        {0X17, 0X17},        //HSTART
        {0X18, 0X05},        //HSTOP
        {0X03, 0X0A},        //VREF
        {0X19, 0X02},        //VSTART
        {0X1A, 0X7A},                //VSTOP

        // SCALLING NUMBERS
        {0X70, 0X3A},        //X_SCALING
        {0X71, 0X35},        //Y_SCALING
        {0X72, 0X11},        //DCW_SCALING
        {0X73, 0XF0},        //PCLK_DIV_SCALING
```

```
{0XA2, 0X02},          //PCLK_DELAY_SCALING

// MATRIX COEFFICIENTS
{0X4F, 0X80},          {0X50, 0X80},
{0X51, 0X00},          {0X52, 0X22},
{0X53, 0X5E},          {0X54, 0X80},
{0X58, 0X9E},

// GAMMA CURVE VALUES
{0X7A, 0X20},          {0X7B, 0X10},
{0X7C, 0X1E},          {0X7D, 0X35},
{0X7E, 0X5A},                  {0X7F, 0X69},
{0X80, 0X76},          {0X81, 0X80},
{0X82, 0X88},          {0X83, 0X8F},
{0X84, 0X96},          {0X85, 0XA3},
{0X86, 0XAF},          {0X87, 0XC4},
{0X88, 0XD7},          {0X89, 0XE8},

// AGC AND AEC PARAMETERS
{0XA5, 0X05},          {0XAB, 0X07},
{0X24, 0X95},          {0X25, 0X33},
{0X26, 0XE3},          {0X9F, 0X78},
{0XA0, 0X68},          {0XA1, 0X03},
{0XA6, 0XD8},                  {0XA7, 0XD8},
{0XA8, 0XF0},          {0XA9, 0X90},
{0XAA, 0X94},                  {0X10, 0X00},

// AWB PARAMETERS
{0X43, 0X0A},          {0X44, 0XF0},
{0X45, 0X34},          {0X46, 0X58},
{0X47, 0X28},          {0X48, 0X3A},
{0X59, 0X88},          {0X5A, 0X88},
{0X5B, 0X44},          {0X5C, 0X67},
{0X5D, 0X49},          {0X5E, 0X0E},
{0X6C, 0X0A},                  {0X6D, 0X55},
{0X6E, 0X11},          {0X6F, 0X9F},
{0X6A, 0X40},          {0X01, 0X40},
{0X02, 0X60},          {0X13, 0XE7},

// ADDITIONAL PARAMETERS
{0X34, 0X11},          {0X3F, 0X00},
{0X75, 0X05},          {0X76, 0XE1},
{0X4C, 0X00},          {0X77, 0X01},
{0XB8, 0X0A},                  {0X41, 0X18},
{0X3B, 0X12},          {0XA4, 0X88},
{0X96, 0X00},          {0X97, 0X30},
{0X98, 0X20},          {0X99, 0X30},
{0X9A, 0X84},          {0X9B, 0X29},
{0X9C, 0X03},          {0X9D, 0X4C},
```

```
        {0X9E, 0X3F},           {0X78, 0X04},
        {0X0E, 0X61},           {0X0F, 0X4B},
        {0X16, 0X02},           {0X1E, 0X00},
        {0X21, 0X02},           {0X22, 0X91},
        {0X29, 0X07},           {0X33, 0X0B},
        {0X35, 0X0B},           {0X37, 0X1D},
        {0X38, 0X71},           {0X39, 0X2A},
        {0X3C, 0X78},           {0X4D, 0X40},
        {0X4E, 0X20},           {0X69, 0X00},
        {0X6B, 0X3A},                   {0X74, 0X10},
        {0X8D, 0X4F},           {0X8E, 0X00},
        {0X8F, 0X00},           {0X90, 0X00},
        {0X91, 0X00},           {0X96, 0X00},
        {0X9A, 0X00},           {0XB1, 0X0C},
        {0XB2, 0X0E},           {0XB3, 0X82},
        {0X4B, 0X01},
};
VOID DELAY(VOLATILE UINT16_T NCOUNT);
VOID DELAY(VOLATILE UINT16_T NCOUNT)
{
  WHILE(NCOUNT--){
  }
}

VOID MCO1_INIT(VOID){
        GPIO_INITTYPEDEF GPIO_INITSTRUCTURE;

        RCC_CLOCKSECURITYSYSTEMCMD(ENABLE);

        RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOA, ENABLE);

        // GPIO CONFIG
        GPIO_INITSTRUCTURE.GPIO_PIN = GPIO_PIN_8;             //PA8 - XCLK
        GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_100MHZ;
        GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
        GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_PP;
        GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP;
        GPIO_INIT(GPIOA, &GPIO_INITSTRUCTURE);

        // GPIO AF CONFIG
        GPIO_PINAFCONFIG(GPIOA, GPIO_PINSOURCE8, GPIO_AF_MCO);

        // MCO CLOCK SOURCE
        RCC_MCO1CONFIG(RCC_MCO1SOURCE_PLLCLK, RCC_MCO1DIV_4);
}

VOID SCCB_INIT(VOID){
        GPIO_INITTYPEDEF GPIO_INITSTRUCTURE;
        I2C_INITTYPEDEF I2C_INITSTRUCTURE;
```

```
 RCC_APB1PERIPHCLOCKCMD(RCC_APB1PERIPH_I2C1, ENABLE);
 RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOB, ENABLE);

        // GPIO CONFIG
        GPIO_INITSTRUCTURE.GPIO_PIN = GPIO_PIN_8 | GPIO_PIN_9;          //PB8 - SIOC


                                //PB9 - SIOD
        GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
        GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_2MHZ;
        GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_OD;
        GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP;
        GPIO_INIT(GPIOB, &GPIO_INITSTRUCTURE);

        // GPIO AF CONFIG
        GPIO_PINAFCONFIG(GPIOB, GPIO_PINSOURCE8, GPIO_AF_I2C1);
        GPIO_PINAFCONFIG(GPIOB, GPIO_PINSOURCE9, GPIO_AF_I2C1);

        // I2C CONFIG
 I2C_DEINIT(I2C1);
 I2C_INITSTRUCTURE.I2C_MODE = I2C_MODE_I2C;
 I2C_INITSTRUCTURE.I2C_DUTYCYCLE = I2C_DUTYCYCLE_2;
 I2C_INITSTRUCTURE.I2C_OWNADDRESS1 = 0X00;
 I2C_INITSTRUCTURE.I2C_ACK = I2C_ACK_ENABLE;
 I2C_INITSTRUCTURE.I2C_ACKNOWLEDGEDADDRESS                                      =
I2C_ACKNOWLEDGEDADDRESS_7BIT;
 I2C_INITSTRUCTURE.I2C_CLOCKSPEED = 100000;
        I2C_ITCONFIG(I2C1, I2C_IT_ERR, ENABLE);
 I2C_INIT(I2C1,&I2C_INITSTRUCTURE);
        I2C_CMD(I2C1, ENABLE);
}

BOOL SCCB_WRITE_REG(UINT8_T REG_ADDR, UINT8_T* DATA);
BOOL SCCB_WRITE_REG(UINT8_T REG_ADDR, UINT8_T* DATA)
{
        UINT32_T TIMEOUT = 0X7FFFFF;

        WHILE(I2C_GETFLAGSTATUS(I2C1, I2C_FLAG_BUSY)){
                IF ((TIMEOUT--) == 0){
                        RETURN TRUE;
                }
        }

        // SEND START BIT
        I2C_GENERATESTART(I2C1, ENABLE);

 WHILE( !I2C_CHECKEVENT(I2C1,I2C_EVENT_MASTER_MODE_SELECT)){
                IF ((TIMEOUT--) == 0){
                        RETURN TRUE;
```

```
                }
  }

        // SEND SLAVE ADDRESS (CAMERA WRITE ADDRESS)
  I2C_SEND7BITADDRESS(I2C1, OV7670_WRITE_ADDR, I2C_DIRECTION_TRANSMITTER);

        WHILE(!I2C_CHECKEVENT(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)){
                IF ((TIMEOUT--) == 0){
                        RETURN TRUE;
                }
  }

        // SEND REGISTER ADDRESS
        I2C_SENDDATA(I2C1, REG_ADDR);

        WHILE(!I2C_CHECKEVENT(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
                IF ((TIMEOUT--) == 0){
                        RETURN TRUE;
                }
  }

        // SEND NEW REGISTER VALUE
  I2C_SENDDATA(I2C1, *DATA);

  WHILE(!I2C_CHECKEVENT(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
                IF ((TIMEOUT--) == 0){
                        RETURN TRUE;
                }
  }

        // SEND STOP BIT
  I2C_GENERATESTOP(I2C1, ENABLE);
        RETURN FALSE;
}

BOOL OV7670_INIT(VOID){
        UINT8_T DATA, I = 0;
        BOOL ERR;

        // CONFIGURE CAMERA REGISTERS
        FOR(I=0; I<OV7670_REG_NUM ;I++){
                DATA = OV7670_REG[I][1];
                ERR = SCCB_WRITE_REG(OV7670_REG[I][0], &DATA);

                IF (ERR == TRUE)
                        BREAK;

                LCD_ILI9341_DRAWLINE(99+I, 110, 99+I, 130, ILI9341_COLOR_WHITE);
```

```
            DELAY(0XFFFF);
        }

        RETURN ERR;
}

VOID DCMI_DMA_INIT(VOID){
        GPIO_INITTYPEDEF GPIO_INITSTRUCTURE;
 DCMI_INITTYPEDEF DCMI_INITSTRUCTURE;
 DMA_INITTYPEDEF  DMA_INITSTRUCTURE;
        NVIC_INITTYPEDEF NVIC_INITSTRUCTURE;

        RCC_AHB2PERIPHCLOCKCMD(RCC_AHB2PERIPH_DCMI, ENABLE);
 RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOA, ENABLE);
        RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOB, ENABLE);
        RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOC, ENABLE);
        RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_GPIOE, ENABLE);
 RCC_AHB1PERIPHCLOCKCMD(RCC_AHB1PERIPH_DMA2, ENABLE);

        // GPIO CONFIG
        GPIO_INITSTRUCTURE.GPIO_PIN = GPIO_PIN_4 | GPIO_PIN_6;              //PA4 - HREF


                                //PA6 - PCLK
 GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
 GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_100MHZ;
 GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_PP;
 GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP ;
 GPIO_INIT(GPIOA, &GPIO_INITSTRUCTURE);

 GPIO_INITSTRUCTURE.GPIO_PIN = GPIO_PIN_6 | GPIO_PIN_7;            //PB6 - D5


                                //PB7 - VSYNC
 GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
 GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_100MHZ;
 GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_PP;
 GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP ;
 GPIO_INIT(GPIOB, &GPIO_INITSTRUCTURE);

 GPIO_INITSTRUCTURE.GPIO_PIN = GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9;
        //PC6 - D0



                                //PC7 - D1
```

```
                                //PC8 - D2


                                //PC9 - D3
  GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
  GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_100MHZ;
  GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_PP;
  GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP ;
  GPIO_INIT(GPIOC, &GPIO_INITSTRUCTURE);

  GPIO_INITSTRUCTURE.GPIO_PIN =  GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6;          //PE4 - D4


                                                                                 //PE5 - D6


                                                                                 //PE6 - D7
  GPIO_INITSTRUCTURE.GPIO_MODE = GPIO_MODE_AF;
  GPIO_INITSTRUCTURE.GPIO_SPEED = GPIO_SPEED_100MHZ;
  GPIO_INITSTRUCTURE.GPIO_OTYPE = GPIO_OTYPE_PP;
  GPIO_INITSTRUCTURE.GPIO_PUPD = GPIO_PUPD_UP ;
  GPIO_INIT(GPIOE, &GPIO_INITSTRUCTURE);

        // GPIO AF CONFIG
        GPIO_PINAFCONFIG(GPIOB, GPIO_PINSOURCE7, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOA, GPIO_PINSOURCE4, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOA, GPIO_PINSOURCE6, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOC, GPIO_PINSOURCE6, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOC, GPIO_PINSOURCE7, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOC, GPIO_PINSOURCE8, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOC, GPIO_PINSOURCE9, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOE, GPIO_PINSOURCE4, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOB, GPIO_PINSOURCE6, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOE, GPIO_PINSOURCE5, GPIO_AF_DCMI);
  GPIO_PINAFCONFIG(GPIOE, GPIO_PINSOURCE6, GPIO_AF_DCMI);

        // DCMI CONFIG
        DCMI_DEINIT();
  DCMI_INITSTRUCTURE.DCMI_CAPTUREMODE = DCMI_CAPTUREMODE_CONTINUOUS;
  DCMI_INITSTRUCTURE.DCMI_EXTENDEDDATAMODE                                       =
DCMI_EXTENDEDDATAMODE_8B;
  DCMI_INITSTRUCTURE.DCMI_CAPTURERATE = DCMI_CAPTURERATE_ALL_FRAME;
        DCMI_INITSTRUCTURE.DCMI_PCKPOLARITY = DCMI_PCKPOLARITY_RISING;
        DCMI_INITSTRUCTURE.DCMI_HSPOLARITY = DCMI_HSPOLARITY_LOW;
  DCMI_INITSTRUCTURE.DCMI_VSPOLARITY = DCMI_VSPOLARITY_HIGH;
  DCMI_INITSTRUCTURE.DCMI_SYNCHROMODE = DCMI_SYNCHROMODE_HARDWARE;
  DCMI_INIT(&DCMI_INITSTRUCTURE);
```

```
	DCMI_CMD(ENABLE);

	// DMA CONFIG
  DMA_DEINIT(DMA2_STREAM1);
	DMA_INITSTRUCTURE.DMA_CHANNEL = DMA_CHANNEL_1;
  DMA_INITSTRUCTURE.DMA_PERIPHERALBASEADDR = (UINT32_T)(&DCMI->DR);
  DMA_INITSTRUCTURE.DMA_MEMORY0BASEADDR = (UINT32_T)FRAME_BUFFER;
  DMA_INITSTRUCTURE.DMA_DIR = DMA_DIR_PERIPHERALTOMEMORY;
  DMA_INITSTRUCTURE.DMA_BUFFERSIZE = IMG_ROWS*IMG_COLUMNS*2/4;
  DMA_INITSTRUCTURE.DMA_PERIPHERALINC = DMA_PERIPHERALINC_DISABLE;
  DMA_INITSTRUCTURE.DMA_MEMORYINC = DMA_MEMORYINC_ENABLE;
  DMA_INITSTRUCTURE.DMA_PERIPHERALDATASIZE                                    =
DMA_PERIPHERALDATASIZE_WORD;
  DMA_INITSTRUCTURE.DMA_MEMORYDATASIZE                                        =
DMA_MEMORYDATASIZE_HALFWORD;
  DMA_INITSTRUCTURE.DMA_MODE = DMA_MODE_NORMAL;
  DMA_INITSTRUCTURE.DMA_PRIORITY = DMA_PRIORITY_HIGH;
  DMA_INITSTRUCTURE.DMA_FIFOMODE = DMA_FIFOMODE_ENABLE;
  DMA_INITSTRUCTURE.DMA_FIFOTHRESHOLD = DMA_FIFOTHRESHOLD_FULL;
  DMA_INITSTRUCTURE.DMA_MEMORYBURST = DMA_MEMORYBURST_SINGLE;
  DMA_INITSTRUCTURE.DMA_PERIPHERALBURST = DMA_PERIPHERALBURST_SINGLE;
  DMA_INIT(DMA2_STREAM1, &DMA_INITSTRUCTURE);
	DMA_CMD(DMA2_STREAM1, ENABLE);

	// DMA INTERRUPT
	DMA_ITCONFIG(DMA2_STREAM1, DMA_IT_TC, ENABLE);

	NVIC_INITSTRUCTURE.NVIC_IRQCHANNEL = DMA2_STREAM1_IRQN;
	NVIC_INITSTRUCTURE.NVIC_IRQCHANNELPREEMPTIONPRIORITY = 0;
	NVIC_INITSTRUCTURE.NVIC_IRQCHANNELSUBPRIORITY = 0;
	NVIC_INITSTRUCTURE.NVIC_IRQCHANNELCMD = ENABLE;
	NVIC_INIT(&NVIC_INITSTRUCTURE);
}
```

Програмні модулі для керування
кольоровим графічним індикатором ILI9341

```
#include "lcd_ili9341.h"

uint16_t ILI9341_x;
uint16_t ILI9341_y;
LCD_ILI931_Options_t ILI9341_Opts;
uint8_t ILI9341_INT_CalledFromPuts = 0;

void LCD_ILI9341_Init() {
	GPIO_InitTypeDef GPIO_InitDef;

	RCC_AHB1PeriphClockCmd(ILI9341_WRX_CLK, ENABLE);
```

```
	GPIO_InitDef.GPIO_Pin = ILI9341_WRX_PIN;
	GPIO_InitDef.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_InitDef.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
	GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(ILI9341_WRX_PORT, &GPIO_InitDef);

	RCC_AHB1PeriphClockCmd(ILI9341_CS_CLK, ENABLE);
	GPIO_InitDef.GPIO_Pin = ILI9341_CS_PIN;
	GPIO_Init(ILI9341_CS_PORT, &GPIO_InitDef);

	RCC_AHB1PeriphClockCmd(ILI9341_RST_CLK, ENABLE);
	GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_InitDef.GPIO_Pin = ILI9341_RST_PIN;
	GPIO_Init(ILI9341_RST_PORT, &GPIO_InitDef);

	ILI9341_CS_SET;

	LCD_SPI_Init();

	LCD_ILI9341_InitLCD();

	ILI9341_x = ILI9341_y = 0;

	ILI9341_Opts.width = ILI9341_WIDTH;
	ILI9341_Opts.height = ILI9341_HEIGHT;
	ILI9341_Opts.orientation = LCD_ILI9341_Portrait;
}

void LCD_ILI9341_InitLCD(void) {
	// ILI9341_RST_RESET;
	ILI9341_RST_SET;

	LCD_ILI9341_SendCommand(ILI9341_RESET);
	LCD_ILI9341_Delay(2000000);

	LCD_ILI9341_SendCommand(ILI9341_POWERA);
	LCD_ILI9341_SendData(0x39);
	LCD_ILI9341_SendData(0x2C);
	LCD_ILI9341_SendData(0x00);
	LCD_ILI9341_SendData(0x34);
	LCD_ILI9341_SendData(0x02);
	LCD_ILI9341_SendCommand(ILI9341_POWERB);
	LCD_ILI9341_SendData(0x00);
	LCD_ILI9341_SendData(0xC1);
	LCD_ILI9341_SendData(0x30);
	LCD_ILI9341_SendCommand(ILI9341_DTCA);
	LCD_ILI9341_SendData(0x85);
	LCD_ILI9341_SendData(0x00);
```

```
LCD_ILI9341_SendData(0x78);
LCD_ILI9341_SendCommand(ILI9341_DTCB);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendCommand(ILI9341_POWER_SEQ);
LCD_ILI9341_SendData(0x64);
LCD_ILI9341_SendData(0x03);
LCD_ILI9341_SendData(0x12);
LCD_ILI9341_SendData(0x81);
LCD_ILI9341_SendCommand(ILI9341_PRC);
LCD_ILI9341_SendData(0x20);
LCD_ILI9341_SendCommand(ILI9341_POWER1);
LCD_ILI9341_SendData(0x23);
LCD_ILI9341_SendCommand(ILI9341_POWER2);
LCD_ILI9341_SendData(0x10);
LCD_ILI9341_SendCommand(ILI9341_VCOM1);
LCD_ILI9341_SendData(0x3E);
LCD_ILI9341_SendData(0x28);
LCD_ILI9341_SendCommand(ILI9341_VCOM2);
LCD_ILI9341_SendData(0x86);
LCD_ILI9341_SendCommand(ILI9341_MAC);
LCD_ILI9341_SendData(0x48);
LCD_ILI9341_SendCommand(ILI9341_PIXEL_FORMAT);
LCD_ILI9341_SendData(0x55);
LCD_ILI9341_SendCommand(ILI9341_FRC);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x18);
LCD_ILI9341_SendCommand(ILI9341_DFC);
LCD_ILI9341_SendData(0x08);
LCD_ILI9341_SendData(0x82);
LCD_ILI9341_SendData(0x27);
LCD_ILI9341_SendCommand(ILI9341_3GAMMA_EN);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendCommand(ILI9341_COLUMN_ADDR);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0xEF);
LCD_ILI9341_SendCommand(ILI9341_PAGE_ADDR);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x00);
LCD_ILI9341_SendData(0x01);
LCD_ILI9341_SendData(0x3F);
LCD_ILI9341_SendCommand(ILI9341_GAMMA);
LCD_ILI9341_SendData(0x01);
LCD_ILI9341_SendCommand(ILI9341_PGAMMA);
LCD_ILI9341_SendData(0x0F);
LCD_ILI9341_SendData(0x31);
LCD_ILI9341_SendData(0x2B);
```

```
	LCD_ILI9341_SendData(0x0C);
	LCD_ILI9341_SendData(0x0E);
	LCD_ILI9341_SendData(0x08);
	LCD_ILI9341_SendData(0x4E);
	LCD_ILI9341_SendData(0xF1);
	LCD_ILI9341_SendData(0x37);
	LCD_ILI9341_SendData(0x07);
	LCD_ILI9341_SendData(0x10);
	LCD_ILI9341_SendData(0x03);
	LCD_ILI9341_SendData(0x0E);
	LCD_ILI9341_SendData(0x09);
	LCD_ILI9341_SendData(0x00);
	LCD_ILI9341_SendCommand(ILI9341_NGAMMA);
	LCD_ILI9341_SendData(0x00);
	LCD_ILI9341_SendData(0x0E);
	LCD_ILI9341_SendData(0x14);
	LCD_ILI9341_SendData(0x03);
	LCD_ILI9341_SendData(0x11);
	LCD_ILI9341_SendData(0x07);
	LCD_ILI9341_SendData(0x31);
	LCD_ILI9341_SendData(0xC1);
	LCD_ILI9341_SendData(0x48);
	LCD_ILI9341_SendData(0x08);
	LCD_ILI9341_SendData(0x0F);
	LCD_ILI9341_SendData(0x0C);
	LCD_ILI9341_SendData(0x31);
	LCD_ILI9341_SendData(0x36);
	LCD_ILI9341_SendData(0x0F);
	LCD_ILI9341_SendCommand(ILI9341_SLEEP_OUT);

	LCD_ILI9341_Delay(1000000);

	LCD_ILI9341_SendCommand(ILI9341_DISPLAY_ON);
	LCD_ILI9341_SendCommand(ILI9341_GRAM);
}


void LCD_ILI9341_SendCommand(uint8_t data) {
	ILI9341_WRX_RESET;
	ILI9341_CS_RESET;
	LCD_SPI_Send(ILI9341_SPI, data);
	ILI9341_CS_SET;
}

void LCD_ILI9341_SendData(uint8_t data) {
	ILI9341_WRX_SET;
	ILI9341_CS_RESET;
	LCD_SPI_Send(ILI9341_SPI, data);
	ILI9341_CS_SET;
```

```
}

void LCD_ILI9341_DrawPixel(uint16_t x, uint16_t y, uint16_t color) {
	LCD_ILI9341_SetCursorPosition(x, y, x, y);

	LCD_ILI9341_SendCommand(ILI9341_GRAM);
	LCD_ILI9341_SendData(color >> 8);
	LCD_ILI9341_SendData(color & 0xFF);
}


void LCD_ILI9341_SetCursorPosition(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2) {
	LCD_ILI9341_SendCommand(ILI9341_COLUMN_ADDR);
	LCD_ILI9341_SendData(x1 >> 8);
	LCD_ILI9341_SendData(x1 & 0xFF);
	LCD_ILI9341_SendData(x2 >> 8);
	LCD_ILI9341_SendData(x2 & 0xFF);

	LCD_ILI9341_SendCommand(ILI9341_PAGE_ADDR);
	LCD_ILI9341_SendData(y1 >> 8);
	LCD_ILI9341_SendData(y1 & 0xFF);
	LCD_ILI9341_SendData(y2 >> 8);
	LCD_ILI9341_SendData(y2 & 0xFF);
}

void LCD_ILI9341_Fill(uint16_t color) {
	unsigned int n, i, j;
	i = color >> 8;
	j = color & 0xFF;
	LCD_ILI9341_SetCursorPosition(0, 0, ILI9341_Opts.width - 1, ILI9341_Opts.height - 1);

	LCD_ILI9341_SendCommand(ILI9341_GRAM);

	for (n = 0; n < ILI9341_PIXEL; n++) {
		LCD_ILI9341_SendData(i);
		LCD_ILI9341_SendData(j);
	}
}

void LCD_ILI9341_DisplayImage(uint16_t image[ILI9341_PIXEL]) {
	uint32_t n, i, j;

	LCD_ILI9341_SetCursorPosition(0, 0, ILI9341_Opts.width - 1, ILI9341_Opts.height - 1);

	LCD_ILI9341_SendCommand(ILI9341_GRAM);

	ILI9341_WRX_SET;
	ILI9341_CS_RESET;
```

```
	for (n = 0; n < ILI9341_PIXEL; n++) {
		i = image[n] >> 8;
		j = image[n] & 0xFF;

		LCD_SPI_Send(ILI9341_SPI, i);
		LCD_SPI_Send(ILI9341_SPI, j);
	}

	ILI9341_CS_SET;
}

void LCD_ILI9341_Delay(volatile unsigned int delay) {
	for (; delay != 0; delay--);
}

void LCD_ILI9341_Rotate(LCD_ILI9341_Orientation_t orientation) {
	LCD_ILI9341_SendCommand(ILI9341_MAC);
	if (orientation == LCD_ILI9341_Orientation_Portrait_1) {
		LCD_ILI9341_SendData(0x58);
	} else if (orientation == LCD_ILI9341_Orientation_Portrait_2) {
		LCD_ILI9341_SendData(0x88);
	} else if (orientation == LCD_ILI9341_Orientation_Landscape_1) {
		LCD_ILI9341_SendData(0x28);
	} else if (orientation == LCD_ILI9341_Orientation_Landscape_2) {
		LCD_ILI9341_SendData(0xE8);
	}

	if (orientation == LCD_ILI9341_Orientation_Portrait_1 || orientation ==
LCD_ILI9341_Orientation_Portrait_2) {
		ILI9341_Opts.width = ILI9341_WIDTH;
		ILI9341_Opts.height = ILI9341_HEIGHT;
		ILI9341_Opts.orientation = LCD_ILI9341_Portrait;
	} else {
		ILI9341_Opts.width = ILI9341_HEIGHT;
		ILI9341_Opts.height = ILI9341_WIDTH;
		ILI9341_Opts.orientation = LCD_ILI9341_Landscape;
	}
}

void LCD_ILI9341_Puts(uint16_t x, uint16_t y, char *str, LCD_FontDef_t *font, uint16_t foreground,
uint16_t background) {
	uint16_t startX = x;

	/* Set X and Y coordinates */
	ILI9341_x = x;
	ILI9341_y = y;

	while (*str) {
		//New line
```

```
		if (*str == '\n') {
			ILI9341_y += font->FontHeight + 1;
			//if after \n is also \r, than go to the left of the screen
			if (*(str + 1) == '\r') {
				ILI9341_x = 0;
				str++;
			} else {
				ILI9341_x = startX;
			}
			str++;
			continue;
		} else if (*str == '\r') {
			str++;
			continue;
		}

		LCD_ILI9341_Putc(ILI9341_x, ILI9341_y, *str++, font, foreground, background);
	}
}

void LCD_ILI9341_GetStringSize(char *str, LCD_FontDef_t *font, uint16_t *width, uint16_t *height) {
	uint16_t w = 0;
	*height = font->FontHeight;
	while (*str++) {
		w += font->FontWidth;
	}
	*width = w;
}

void LCD_ILI9341_Putc(uint16_t x, uint16_t y, char c, LCD_FontDef_t *font, uint16_t foreground,
uint16_t background) {
	uint32_t i, b, j;
	/* Set coordinates */
	ILI9341_x = x;
	ILI9341_y = y;
	if ((ILI9341_x + font->FontWidth) > ILI9341_Opts.width) {
		//If at the end of a line of display, go to new line and set x to 0 position
		ILI9341_y += font->FontHeight;
		ILI9341_x = 0;
	}
	for (i = 0; i < font->FontHeight; i++) {
		b = font->data[(c - 32) * font->FontHeight + i];
		for (j = 0; j < font->FontWidth; j++) {
			if ((b << j) & 0x8000) {
				LCD_ILI9341_DrawPixel(ILI9341_x + j, (ILI9341_y + i), foreground);
			} else {
				LCD_ILI9341_DrawPixel(ILI9341_x + j, (ILI9341_y + i), background);
			}
		}
```

```
        }
        ILI9341_x += font->FontWidth;
}


void LCD_ILI9341_DrawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color) {
        /* Code by dewoller: https://github.com/dewoller */

        int16_t dx, dy, sx, sy, err, e2;

        /* Check for overflow */
        if (x0 >= ILI9341_Opts.width) {
                x0 = ILI9341_Opts.width - 1;
        }
        if (x1 >= ILI9341_Opts.width) {
                x1 = ILI9341_Opts.width - 1;
        }
        if (y0 >= ILI9341_Opts.height) {
                y0 = ILI9341_Opts.height - 1;
        }
        if (y1 >= ILI9341_Opts.height) {
                y1 = ILI9341_Opts.height - 1;
        }

        dx = (x0 < x1) ? (x1 - x0) : (x0 - x1);
        dy = (y0 < y1) ? (y1 - y0) : (y0 - y1);
        sx = (x0 < x1) ? 1 : -1;
        sy = (y0 < y1) ? 1 : -1;
        err = ((dx > dy) ? dx : -dy) / 2;

        while (1) {
                LCD_ILI9341_DrawPixel(x0, y0, color);
                if (x0 == x1 && y0 == y1) {
                        break;
                }
                e2 = err;
                if (e2 > -dx) {
                        err -= dy;
                        x0 += sx;
                }
                if (e2 < dy) {
                        err += dx;
                        y0 += sy;
                }
        }
}

void LCD_ILI9341_DrawRectangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color) {
        LCD_ILI9341_DrawLine(x0, y0, x1, y0, color); //Top
```

```
	LCD_ILI9341_DrawLine(x0, y0, x0, y1, color);//Left
	LCD_ILI9341_DrawLine(x1, y0, x1, y1, color);//Right
	LCD_ILI9341_DrawLine(x0, y1, x1, y1, color);//Bottom
}

void LCD_ILI9341_DrawFilledRectangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t
color) {
	for (; y0 < y1; y0++) {
		LCD_ILI9341_DrawLine(x0, y0, x1, y0, color);
	}
}

void LCD_ILI9341_DrawCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color) {
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
	int16_t ddF_y = -2 * r;
	int16_t x = 0;
	int16_t y = r;

   LCD_ILI9341_DrawPixel(x0, y0 + r, color);
   LCD_ILI9341_DrawPixel(x0, y0 - r, color);
   LCD_ILI9341_DrawPixel(x0 + r, y0, color);
   LCD_ILI9341_DrawPixel(x0 - r, y0, color);

   while (x < y) {
      if (f >= 0) {
         y--;
         ddF_y += 2;
         f += ddF_y;
      }
      x++;
      ddF_x += 2;
      f += ddF_x;

      LCD_ILI9341_DrawPixel(x0 + x, y0 + y, color);
      LCD_ILI9341_DrawPixel(x0 - x, y0 + y, color);
      LCD_ILI9341_DrawPixel(x0 + x, y0 - y, color);
      LCD_ILI9341_DrawPixel(x0 - x, y0 - y, color);

      LCD_ILI9341_DrawPixel(x0 + y, y0 + x, color);
      LCD_ILI9341_DrawPixel(x0 - y, y0 + x, color);
      LCD_ILI9341_DrawPixel(x0 + y, y0 - x, color);
      LCD_ILI9341_DrawPixel(x0 - y, y0 - x, color);
   }
}

void LCD_ILI9341_DrawFilledCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color) {
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
```

```
        int16_t ddF_y = -2 * r;
        int16_t x = 0;
        int16_t y = r;

    LCD_ILI9341_DrawPixel(x0, y0 + r, color);
    LCD_ILI9341_DrawPixel(x0, y0 - r, color);
    LCD_ILI9341_DrawPixel(x0 + r, y0, color);
    LCD_ILI9341_DrawPixel(x0 - r, y0, color);
    LCD_ILI9341_DrawLine(x0 - r, y0, x0 + r, y0, color);

    while (x < y) {
      if (f >= 0) {
        y--;
        ddF_y += 2;
        f += ddF_y;
      }
      x++;
      ddF_x += 2;
      f += ddF_x;

      LCD_ILI9341_DrawLine(x0 - x, y0 + y, x0 + x, y0 + y, color);
      LCD_ILI9341_DrawLine(x0 + x, y0 - y, x0 - x, y0 - y, color);

      LCD_ILI9341_DrawLine(x0 + y, y0 + x, x0 - y, y0 + x, color);
      LCD_ILI9341_DrawLine(x0 + y, y0 - x, x0 - y, y0 - x, color);
    }
}



//#include "lcd_ili9341.h"


#IFNDEF LCD_ILI9341_H
#DEFINE LCD_ILI9341_H

// INCLUDES
#INCLUDE "STM32F4XX.H"
#INCLUDE "STM32F4XX_RCC.H"
#INCLUDE "STM32F4XX_GPIO.H"
#INCLUDE "LCD_SPI.H"
#INCLUDE "LCD_FONTS.H"

// SPI SET
#DEFINE ILI9341_SPI                                    SPI5

// CS PIN
#DEFINE ILI9341_CS_CLK                         RCC_AHB1PERIPH_GPIOC
#DEFINE ILI9341_CS_PORT                        GPIOC
```

```
#DEFINE ILI9341_CS_PIN                  GPIO_PIN_2

// WRX PIN
#DEFINE ILI9341_WRX_CLK                 RCC_AHB1PERIPH_GPIOD
#DEFINE ILI9341_WRX_PORT                GPIOD
#DEFINE ILI9341_WRX_PIN                 GPIO_PIN_13

// RESET PIN
#DEFINE ILI9341_RST_CLK                 RCC_AHB1PERIPH_GPIOD
#DEFINE ILI9341_RST_PORT                GPIOD
#DEFINE ILI9341_RST_PIN                 GPIO_PIN_12

#DEFINE ILI9341_RST_SET                 GPIO_SETBITS(ILI9341_RST_PORT,
ILI9341_RST_PIN)
#DEFINE ILI9341_RST_RESET               GPIO_RESETBITS(ILI9341_RST_PORT,
ILI9341_RST_PIN)
#DEFINE ILI9341_CS_SET                  GPIO_SETBITS(ILI9341_CS_PORT,
ILI9341_CS_PIN)
#DEFINE ILI9341_CS_RESET            GPIO_RESETBITS(ILI9341_CS_PORT,
ILI9341_CS_PIN)
#DEFINE ILI9341_WRX_SET                 GPIO_SETBITS(ILI9341_WRX_PORT,
ILI9341_WRX_PIN)
#DEFINE ILI9341_WRX_RESET               GPIO_RESETBITS(ILI9341_WRX_PORT,
ILI9341_WRX_PIN)

// LCD SETTINGS
#DEFINE ILI9341_WIDTH                   240
#DEFINE ILI9341_HEIGHT                  320
#DEFINE ILI9341_PIXEL                   76800

// COLORS
#DEFINE ILI9341_COLOR_WHITE             0XFFFF
#DEFINE ILI9341_COLOR_BLACK             0X0000
#DEFINE ILI9341_COLOR_RED                   0XF800
#DEFINE ILI9341_COLOR_GREEN             0X07E0
#DEFINE ILI9341_COLOR_GREEN2        0XB723
#DEFINE ILI9341_COLOR_BLUE              0X001F
#DEFINE ILI9341_COLOR_BLUE2             0X051D
#DEFINE ILI9341_COLOR_YELLOW            0XFFE0
#DEFINE ILI9341_COLOR_ORANGE            0XFBE4
#DEFINE ILI9341_COLOR_CYAN              0X07FF
#DEFINE ILI9341_COLOR_MAGENTA           0XA254
#DEFINE ILI9341_COLOR_GRAY              0X7BEF //1111 0111 1101 1110
#DEFINE ILI9341_COLOR_BROWN             0XBBCA

// COMMANDS
#DEFINE ILI9341_RESET                       0X01
#DEFINE ILI9341_SLEEP_OUT               0X11
#DEFINE ILI9341_GAMMA                       0X26
```

```
#DEFINE ILI9341_DISPLAY_OFF		0X28
#DEFINE ILI9341_DISPLAY_ON		0X29
#DEFINE ILI9341_COLUMN_ADDR		0X2A
#DEFINE ILI9341_PAGE_ADDR			0X2B
#DEFINE ILI9341_GRAM				0X2C
#DEFINE ILI9341_MAC					0X36
#DEFINE ILI9341_PIXEL_FORMAT  0X3A
#DEFINE ILI9341_WDB					0X51
#DEFINE ILI9341_WCD					0X53
#DEFINE ILI9341_RGB_INTERFACE		0XB0
#DEFINE ILI9341_FRC					0XB1
#DEFINE ILI9341_BPC					0XB5
#DEFINE ILI9341_DFC					0XB6
#DEFINE ILI9341_POWER1			0XC0
#DEFINE ILI9341_POWER2			0XC1
#DEFINE ILI9341_VCOM1				0XC5
#DEFINE ILI9341_VCOM2				0XC7
#DEFINE ILI9341_POWERA			0XCB
#DEFINE ILI9341_POWERB			0XCF
#DEFINE ILI9341_PGAMMA			0XE0
#DEFINE ILI9341_NGAMMA			0XE1
#DEFINE ILI9341_DTCA				0XE8
#DEFINE ILI9341_DTCB				0XEA
#DEFINE ILI9341_POWER_SEQ		0XED
#DEFINE ILI9341_3GAMMA_EN		0XF2
#DEFINE ILI9341_INTERFACE		0XF6
#DEFINE ILI9341_PRC					0XF7

// SELECT ORIENTATION FOR LCD
TYPEDEF ENUM {
	LCD_ILI9341_ORIENTATION_PORTRAIT_1,
	LCD_ILI9341_ORIENTATION_PORTRAIT_2,
	LCD_ILI9341_ORIENTATION_LANDSCAPE_1,
	LCD_ILI9341_ORIENTATION_LANDSCAPE_2
} LCD_ILI9341_ORIENTATION_T;

// ORIENTATION, USED PRIVATE
TYPEDEF ENUM {
	LCD_ILI9341_LANDSCAPE,
	LCD_ILI9341_PORTRAIT
} LCD_ILI9341_ORIENTATION;

// LCD OPTIONS, USED PRIVATE
TYPEDEF STRUCT {
	UINT16_T WIDTH;
	UINT16_T HEIGHT;
	LCD_ILI9341_ORIENTATION ORIENTATION; // 1 = PORTRAIT; 0 = LANDSCAPE
} LCD_ILI931_OPTIONS_T;
```

```
// SELECT FONT
EXTERN LCD_FONTDEF_T LCD_FONT_7X10;
EXTERN LCD_FONTDEF_T LCD_FONT_11X18;
EXTERN LCD_FONTDEF_T LCD_FONT_16X26;

/*
* INITIALIZE ILI9341 LCD
*/
EXTERN VOID LCD_ILI9341_INIT(VOID);

/*
* INIT LCD PINS
*
* CALLED PRIVATE
*/
EXTERN VOID LCD_ILI9341_INITLCD(VOID);

/*
* SEND DATA TO LCD VIA SPI
*
* CALLED PRIVATE
*
* PARAMETERS:
*         - UINT8_T DATA: DATA TO BE SENT
*/
EXTERN VOID LCD_ILI9341_SENDDATA(UINT8_T DATA);

/*
* SEND COMMAND TO LCD VIA SPI
*
* CALLED PRIVATE
*
* PARAMETERS:
*         - UINT8_T DATA: DATA TO BE SENT
*/
EXTERN VOID LCD_ILI9341_SENDCOMMAND(UINT8_T DATA);

/*
* SIMPLE DELAY
*
* PARAMETERS:
*         - VOLATILE UNSIGNED INT DELAY: CLOCK CYCLES
*/
EXTERN VOID LCD_ILI9341_DELAY(VOLATILE UNSIGNED INT DELAY);

/*
* SET CURSOR POSITION
*
* CALLED PRIVATE
```

```
*/
EXTERN VOID LCD_ILI9341_SETCURSORPOSITION(UINT16_T X1, UINT16_T Y1, UINT16_T
X2, UINT16_T Y2);

/*
* DRAW SINGLE PIXEL TO LCD
*
* PARAMETERS:
*       - UINT16_T X: X POSITION FOR PIXEL
*       - UINT16_T Y: Y POSITION FOR PIXEL
*       - UINT16_T COLOR: COLOR OF PIXEL
*/
EXTERN VOID LCD_ILI9341_DRAWPIXEL(UINT16_T X, UINT16_T Y, UINT16_T COLOR);

/*
* FILL ENTIRE LCD WITH COLOR
*
* PARAMETERS:
*       - UINT16_T COLOR: COLOR TO BE USED IN FILL
*/
EXTERN VOID LCD_ILI9341_FILL(UINT16_T COLOR);

/*
* SHOW SELECTED QVGA IMAGE ON ENTIRE LCD
*
* PARAMETERS:
*       - UINT16_T IMAGE: ARRAY OF SELECTED IMAGE IN RB565 FORMAT
*/
EXTERN VOID LCD_ILI9341_DISPLAYIMAGE(UINT16_T IMAGE[ILI9341_PIXEL]);

/*
* ROTATE LCD
* SELECT ORIENTATION
*
* PARAMETERS:
*       - LCD_ILI9341_ORIENTATION_T ORIENTATION
*               - LCD_ILI9341_ORIENTATION_PORTRAIT_1: NO ROTATION
*               - LCD_ILI9341_ORIENTATION_PORTRAIT_2: ROTATE 180DEG
*               - LCD_ILI9341_ORIENTATION_LANDSCAPE_1: ROTATE 90DEG
*               - LCD_ILI9341_ORIENTATION_LANDSCAPE_2: ROTATE -90DEG
*/
EXTERN VOID LCD_ILI9341_ROTATE(LCD_ILI9341_ORIENTATION_T ORIENTATION);

/*
* PUT SINGLE CHARACTER TO LCD
*
* PARAMETERS:
*       - UINT16_T X: X POSITION OF TOP LEFT CORNER
*       - UINT16_T Y: Y POSITION OF TOP LEFT CORNER
```

```
*       - CHAR C: CHARACTER TO BE DISPLAYED
*       - LCD_FONTDEF_T *FONT: POINTER TO USED FONT
*       - UINT16_T FOREGROUND: COLOR FOR CHAR
*       - UINT16_T BACKGROUND: COLOR FOR CHAR BACKGROUND
*/
EXTERN VOID LCD_ILI9341_PUTC(UINT16_T X, UINT16_T Y, CHAR C, LCD_FONTDEF_T
*FONT, UINT16_T FOREGROUND, UINT16_T BACKGROUND);

/*
* PUT STRING TO LCD
*
* PARAMETERS:
*       - UINT16_T X: X POSITION OF TOP LEFT CORNER OF FIRST CHARACTER IN STRING
*       - UINT16_T Y: Y POSITION OF TOP LEFT CORNER OF FIRST CHARACTER IN STRING
*       - CHAR *STR: POINTER TO FIRST CHARACTER
*       - LCD_FONTDEF_T *FONT: POINTER TO USED FONT
*       - UINT16_T FOREGROUND: COLOR FOR STRING
*       - UINT16_T BACKGROUND: COLOR FOR STRING BACKGROUND
*/
EXTERN VOID LCD_ILI9341_PUTS(UINT16_T X, UINT16_T Y, CHAR *STR, LCD_FONTDEF_T
*FONT, UINT16_T FOREGROUND, UINT16_T BACKGROUND);

/*
* GET WIDTH AND HEIGHT OF BOX WITH TEXT
*
* PARAMETERS:
*       - CHAR *STR: POINTER TO FIRST CHARACTER
*       - LCD_FONTDEF_T *FONT: POINTER TO USED FONT
*       - UINT16_T *WIDTH: POINTER TO VARIABLE TO STORE WIDTH
*       - UINT16_T *HEIGHT: OINTER TO VARIABLE TO STORE HEIGHT
*/
EXTERN VOID LCD_ILI9341_GETSTRINGSIZE(CHAR *STR, LCD_FONTDEF_T *FONT,
UINT16_T *WIDTH, UINT16_T *HEIGHT);

/*
* DRAW LINE TO LCD
*
* PARAMETERS:
*       - UINT16_T X0: X COORDINATE OF STARTING POINT
*       - UINT16_T Y0: Y COORDINATE OF STARTING POINT
*       - UINT16_T X1: X COORDINATE OF ENDING POINT
*       - UINT16_T Y1: Y COORDINATE OF ENDING POINT
*       - UINT16_T COLOR: LINE COLOR
*/
EXTERN VOID LCD_ILI9341_DRAWLINE(UINT16_T X0, UINT16_T Y0, UINT16_T X1, UINT16_T
Y1, UINT16_T COLOR);

/*
* DRAW RECTANGLE ON LCD
```

```
*
* PARAMETERS:
* - UINT16_T X0: X COORDINATE OF TOP LEFT POINT
* - UINT16_T Y0: Y COORDINATE OF TOP LEFT POINT
* - UINT16_T X1: X COORDINATE OF BOTTOM RIGHT POINT
* - UINT16_T Y1: Y COORDINATE OF BOTTOM RIGHT POINT
* - UINT16_T COLOR: RECTANGLE COLOR
*/
EXTERN VOID LCD_ILI9341_DRAWRECTANGLE(UINT16_T X0, UINT16_T Y0, UINT16_T X1,
UINT16_T Y1, UINT16_T COLOR);

/*
* DRAW FILLED RECTANGLE ON LCD
*
* PARAMETERS:
* - UINT16_T X0: X COORDINATE OF TOP LEFT POINT
* - UINT16_T Y0: Y COORDINATE OF TOP LEFT POINT
* - UINT16_T X1: X COORDINATE OF BOTTOM RIGHT POINT
* - UINT16_T Y1: Y COORDINATE OF BOTTOM RIGHT POINT
* - UINT16_T COLOR: RECTANGLE COLOR
*/
EXTERN VOID LCD_ILI9341_DRAWFILLEDRECTANGLE(UINT16_T X0, UINT16_T Y0,
UINT16_T X1, UINT16_T Y1, UINT16_T COLOR);

/*
* DRAW CIRCLE ON LCD
*
* PARAMETERS:
* - INT16_T X0: X COORDINATE OF CENTER CIRCLE POINT
* - INT16_T Y0: Y COORDINATE OF CENTER CIRCLE POINT
* - INT16_T R: CIRCLE RADIUS
* - UINT16_T COLOR: CIRCLE COLOR
*/
EXTERN VOID LCD_ILI9341_DRAWCIRCLE(INT16_T X0, INT16_T Y0, INT16_T R, UINT16_T
COLOR);

/*
* DRAW FILLED ON LCD
*
* PARAMETERS:
* - INT16_T X0: X COORDINATE OF CENTER CIRCLE POINT
* - INT16_T Y0: Y COORDINATE OF CENTER CIRCLE POINT
* - INT16_T R: CIRCLE RADIUS
* - UINT16_T COLOR: CIRCLE COLOR
*/
EXTERN VOID LCD_ILI9341_DRAWFILLEDCIRCLE(INT16_T X0, INT16_T Y0, INT16_T R,
UINT16_T COLOR);

#ENDIF
```

Лабораторна робота № 4

ДОСЛІДЖЕННЯ ВУЗЛА ВВЕДЕННЯ В МПС ІНФОРМАЦІЇ З ЦИФРОВОГО СЕНСОРА

МЕТА РОБОТИ:ознайомлення з технічними характеристиками, архітектурою мікроконтролера з **Atmel SAM4E16E з ядром Cortex-M4F**, інтерфейсом TWI/I2C , сенсором температури DS1621 та основними режимами роботи інтегрованої системи **Atmel Studio**.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з структурою мікроконтролера **ATSAM4E16E та модуля SAM4E Xplained Pro Evaluation Kit**
2. Ознайомитися з інтерфейсом TWI/I2C
3. Ознайомитися з сенсором температури DS1621
4. Перевірити свою теоретичну підготовку по контрольних питаннях для самоперевірки; при необхідності скористатися методичними матеріалами.
5. Запустити систему Atmel Studio.
6. Ознайомитись з елементами екрану відлагоджувача, використанням команд головного меню та підменю, режимами відображення завантаженої програми, операціями модифікації вмісту комірок пам'яті та регістрів.
7. Виконати основні етапи процесу налагодження програми в системі Atmel Studio:

– створити новий проєкт або завантажити тестовий приклад

– відкомпілювати проєкт (**Build Solution**).

– запустити режим відлагодження з використанням модуля **SAM4E Xplained Pro Evaluation Kit**

8. Виконати індивідуальне завдання визначене керівником лабораторної роботи.
9. Захистити лабораторну роботу.

ЗАВДАННЯ

1. В покроковому режимі продемонструвати ініціалізацію мікроконтролера та відлагоджувальної плати.
2. В покроковому режимі продемонструвати ініціалізацію портів.
3. В покроковому режимі продемонструвати ініціалізацію TWI/I2C.
4. В покроковому режимі продемонструвати роботу сенсора температури DS1621.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Внутрішня структура мікроконтролера **ATSAM4E16E**.
2. Програмна модель **ATSAM4E16E**.
3. Структура **SAM4E Xplained Pro Evaluation Kit**
4. Особливості організації інтерфейса TWI/I2C
5. Основні опції меню відлагоджувача **Microchip Studio for AVR® and SAM Devices**
6. Структура, контакти, підключення сенсора температури DS1621

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

1. Запустити систему **Microchip Studio for AVR® and SAM Devices**.
2. Ознайомитись з елементами екрану відлагоджувача, використанням команд головного меню та підменю, режимами відображення завантаженої програми, операціями модифікації вмісту комірок пам'яті та регістрів.

Виконати основні етапи процесу налагодження програми в системі **Microchip Studio for AVR® and SAM Devices**:

- створити новий проєкт (**New Project..**) або завантажити приклад (**New Example Project..**) чи попередній готовий проєкт (**Open Project..**).
- відкомпілювати проєкт (**Build Solution**).
- запустити режим відлагодження з використанням модуля **SAM4E Xplained Pro Evaluation Kit**
- налагодити програму у різних режимах командою **Debug**.
- ознайомитися за допомогою інтерактивної довідки, яка входить у склад програмного пакету, з можливостями середовища та його функціями.

3. Ознайомитися з елементи графічного інтерфейсу та з режимами відлагодження. Елементи графічного інтерфейсу коротко описані на рис. 1 – 8.

ОСНОВНІ ЕТАПИ
створення проєкту та налагодження програми в інтегрованій системі Microchip Studio for AVR® and SAM Devices з використанням Starter Kit SAM4E-Xplained-Pro

Запускаємо **Microchip Studio for AVR® and SAM Devices**

*Рис. 4.1. Вікно завантаження **Microchip Studio for AVR® and SAM Devices***

Створюємо новий проєкт (**New Project..**) або завантажуємо приклад (**New Example Project..**) чи попередній готовий проєкт (**Open Project..**).

New Example Project..:

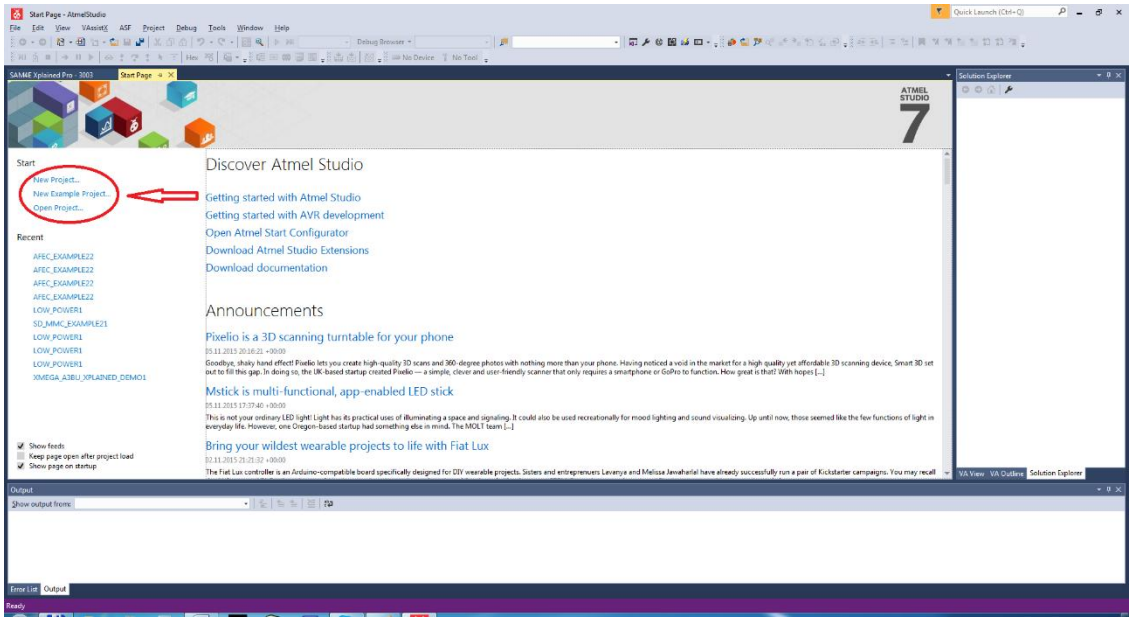


Рис. 4.2. Вибір типу проєкту

Вибираємо приклад програми для Starter Kit SAM4E-Xplained-Pro (SAM4E-XPRO) та каталог для її розміщення:

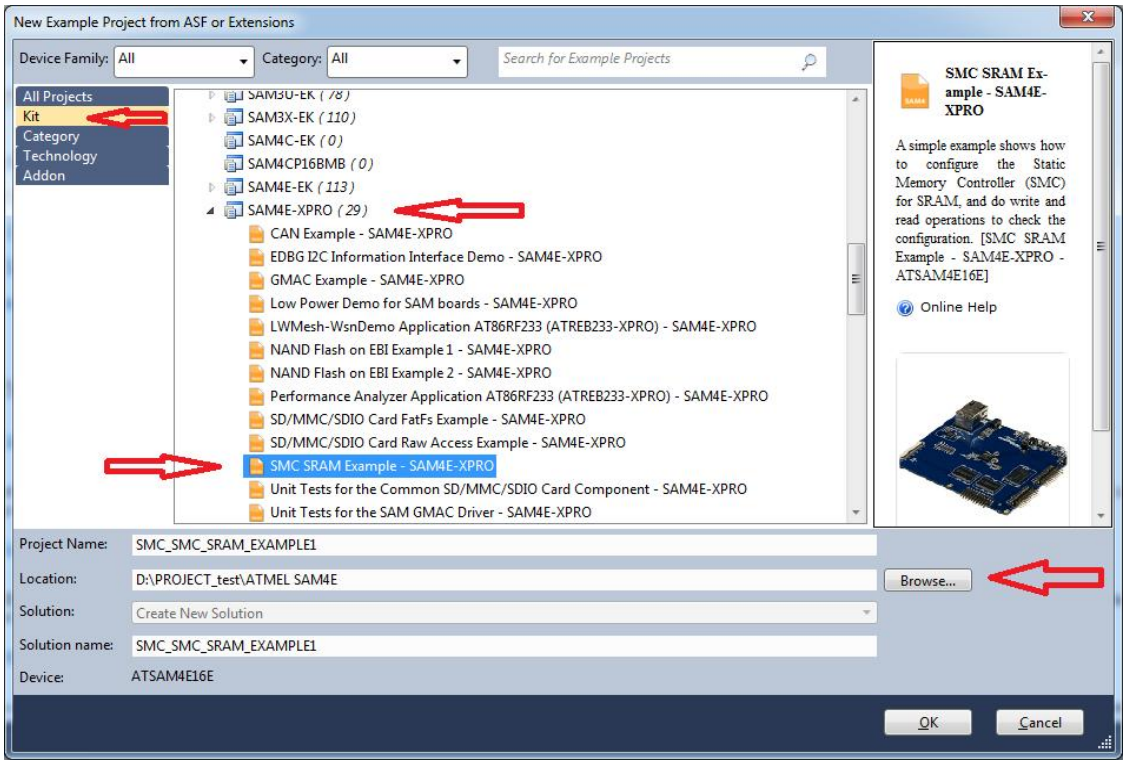


Рис. 4.3. Вибір програми

Підтверджуємо ліцензію використання, натискаємо “**Finish**”, чекаємо на відкривання проєкту, відкриваємо файл програми “**main**” та натискаємо “**Build Solution**”:

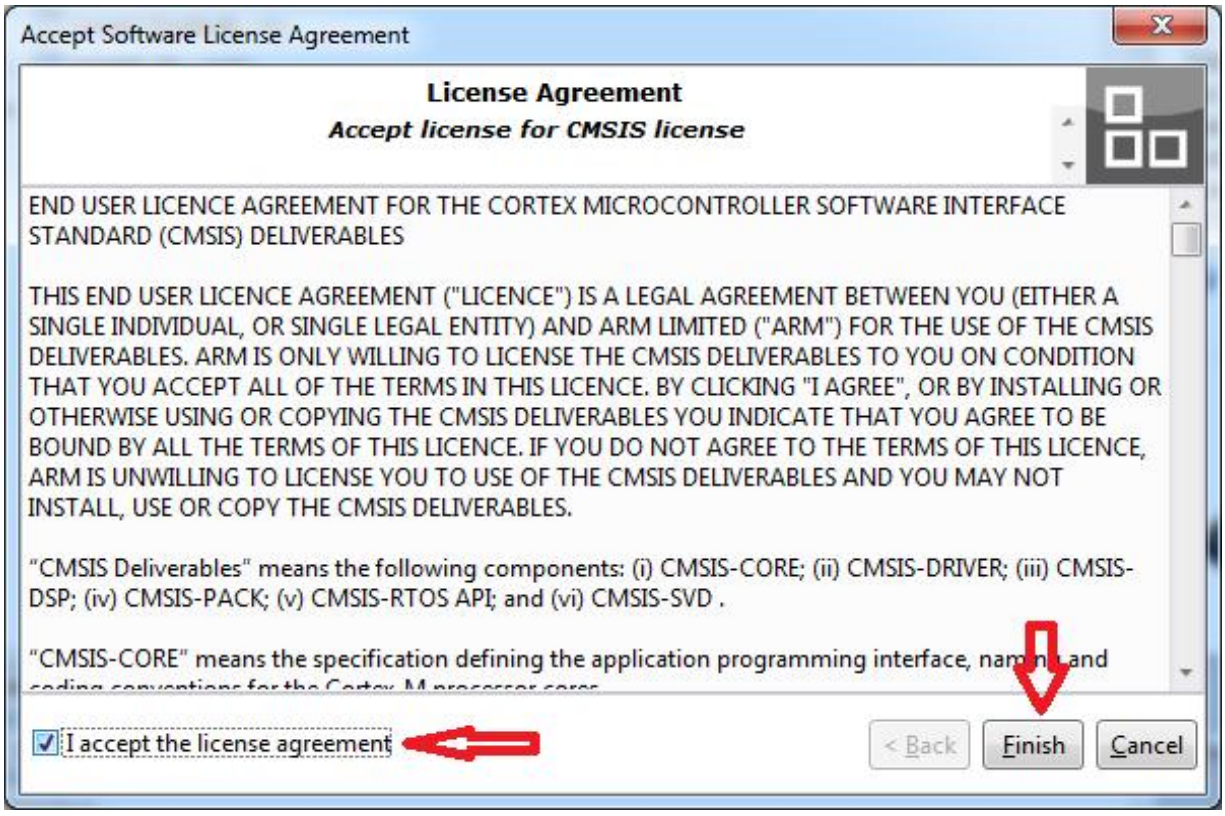


Рис. 4.4. Підтвердження ліцензії

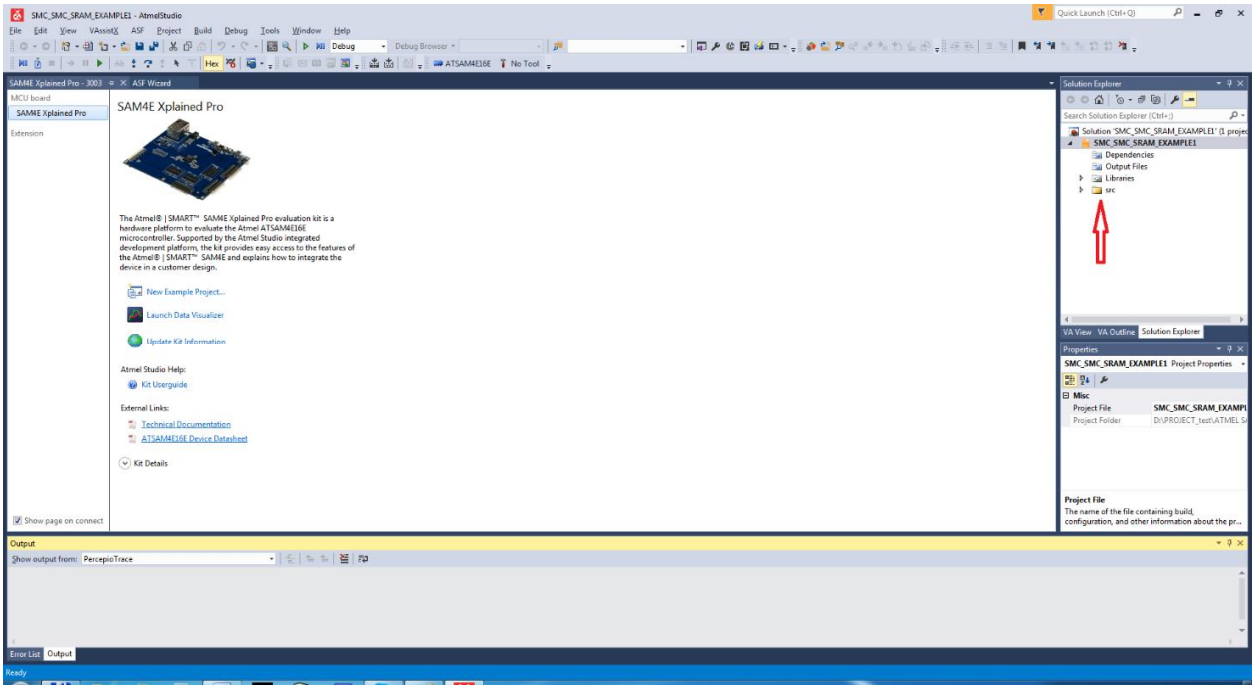


Рис. 4.5. Відкривання файлу “main”

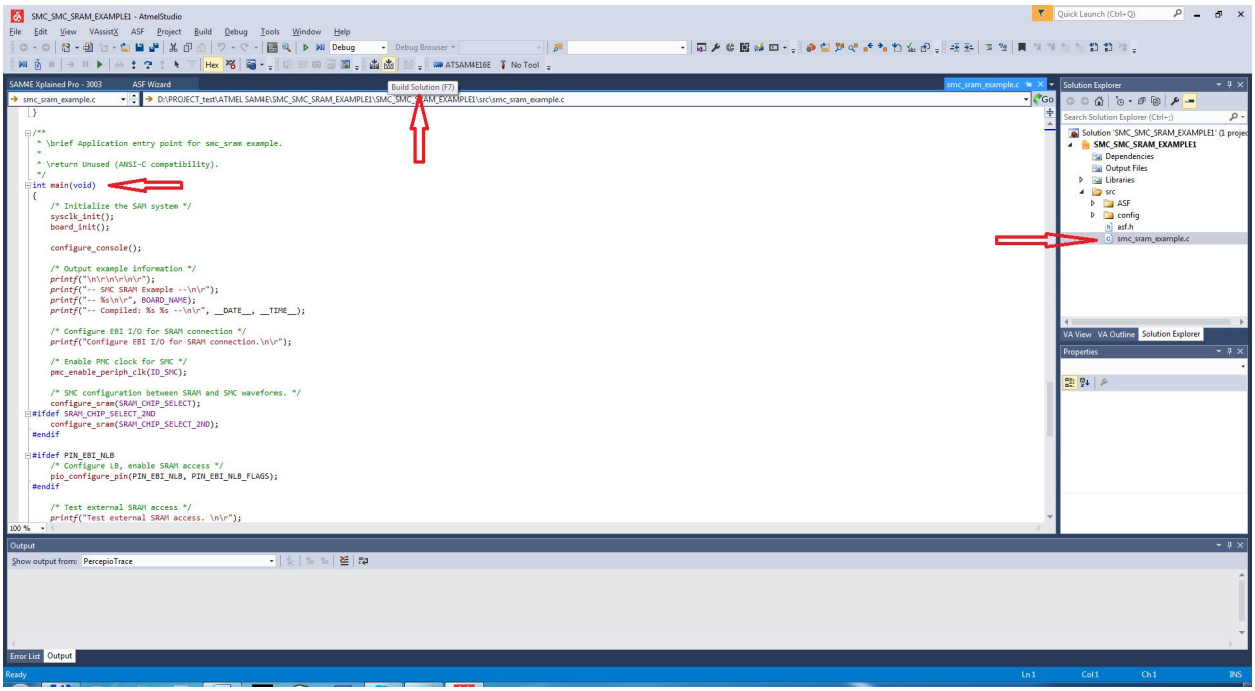


Рис. 4.6. Компіляція програми

Компіляцію та виправлення програми виконуємо до відсутності помилок:

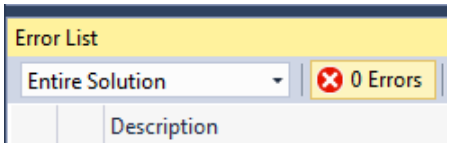


Рис. 4.7. Вікно помилок

При відсутності помилок запускаємо відлагоджувач попередньо підключивши “**Debug USB**” плати **SAM4E-Xplained-Pro** до роз’єму USB ПК:

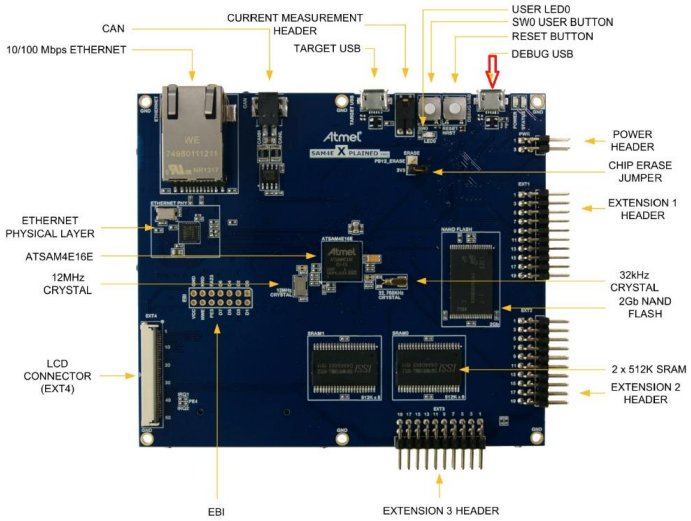


Рис. 4.8. Підключення “Debug USB” плати SAM4E-Xplained-Pro до роз’єму USB ПК

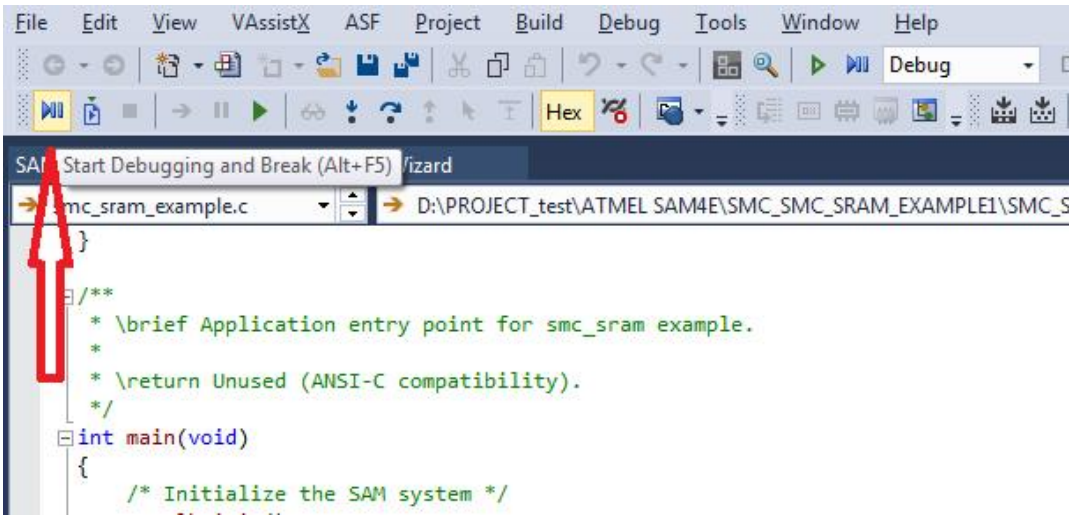


Рис. 4.9а. Запуск відлагодження

Після ПЕРШОГО запуску проєкту появиться повідомлення і натискаємо “Continue”:

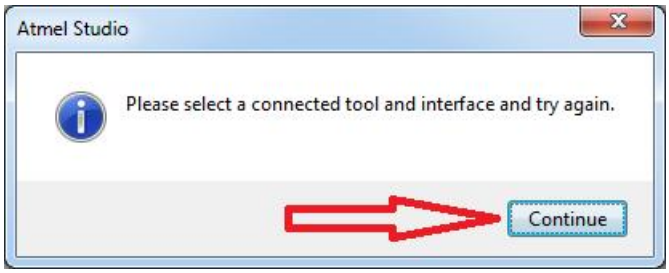


Рис. 4.9б. Запуск відлагодження

Вибираємо тип відлагоджувача/програматора:

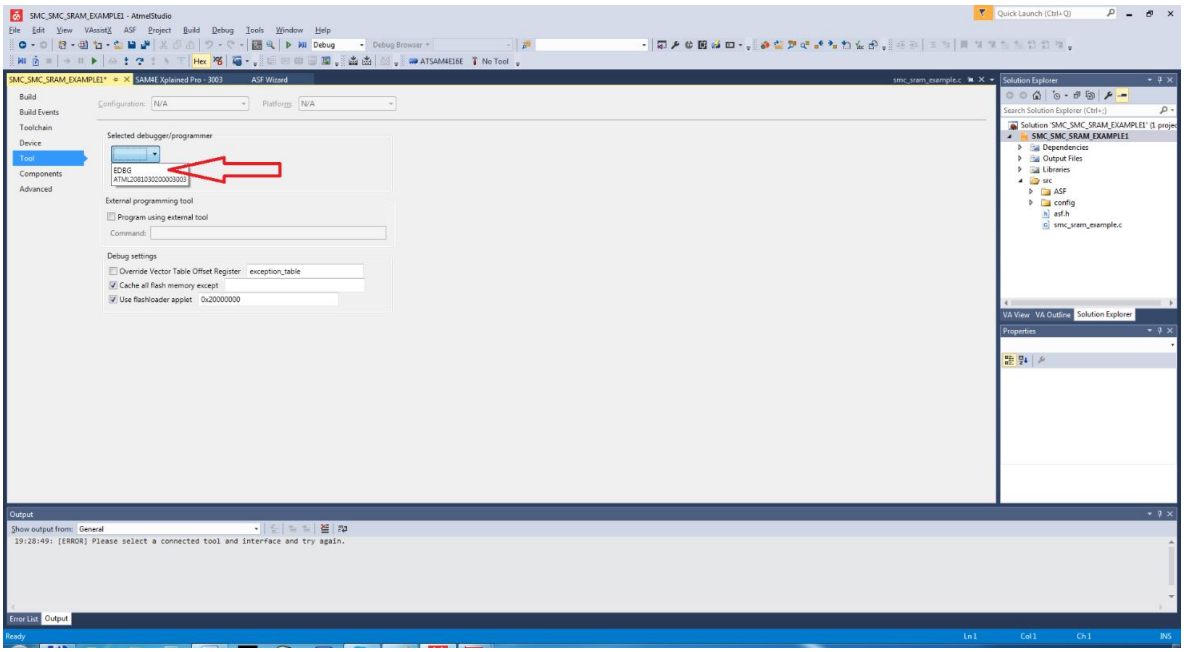


Рис. 4.10а. Вибір типу відлагоджувача/програматора

Появиться тип відлагоджувача/програматора та тип інтерфесу і знову запускаємо відлагоджувач:

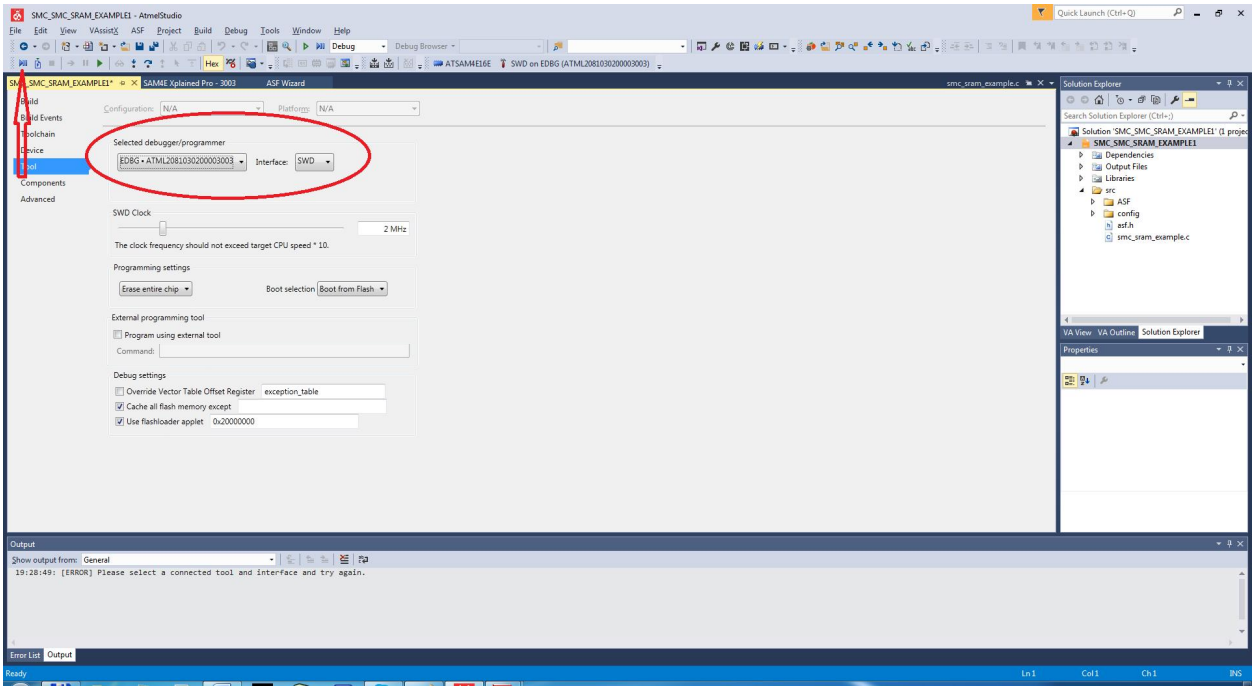


Рис. 4.10б. Вибір типу відлагоджувача/програматора

Після успішного запуску вибираємо програму “main”:

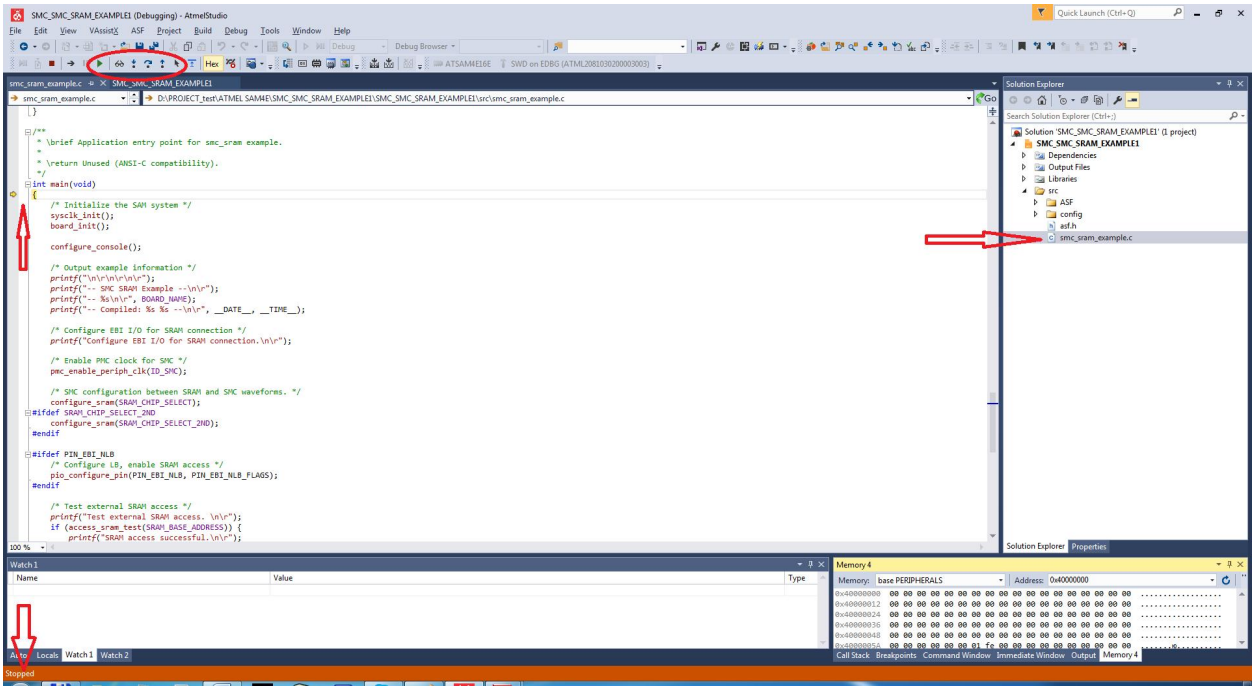


Рис. 4.11. Запуск відлагодження в різних режимах

Після цього можна запустити виконання програми в різних режимах та контролювати її виконання.

ОСНОВНІ ЕТАПИ
створення проєкту та налагодження програми
в інтегрованому середовищі Microchip Studio for AVR® and SAM Devices
в режимі New Project..

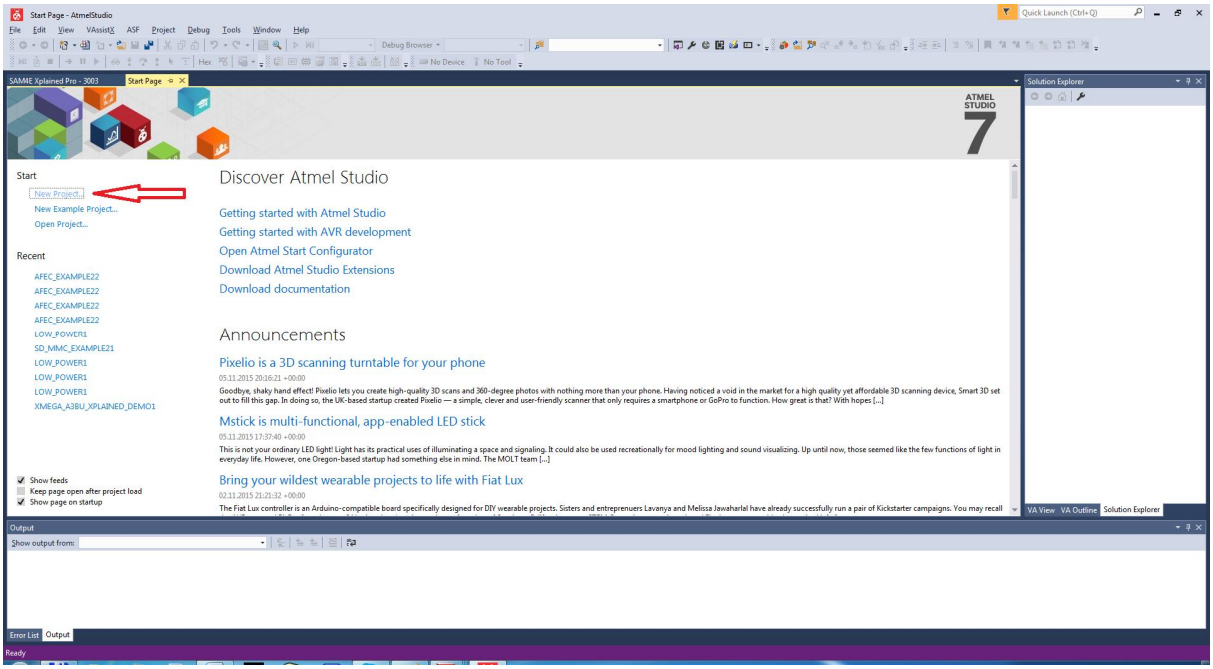


*Рис. 4.12. Створення нового проєкту (**New Project..**)*

Вибираємо тип проєкту, каталог розміщення та ім.'я проєкту

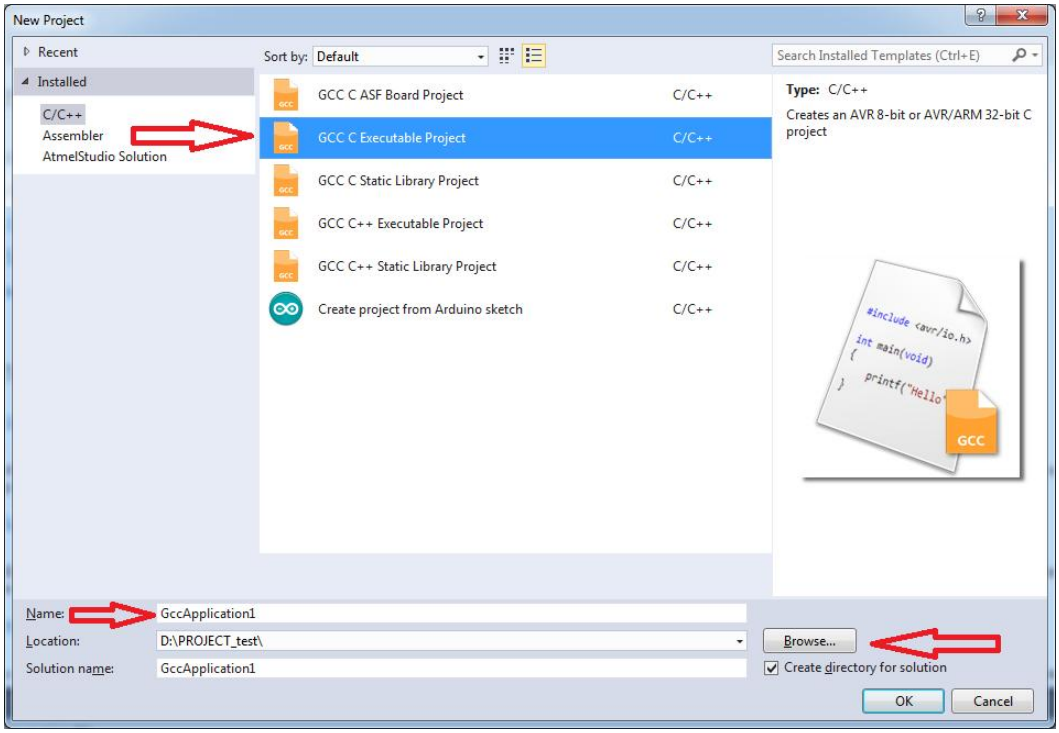


Рис. 4.13. Вибір типу проєкту

Вибираємо тип процесора:

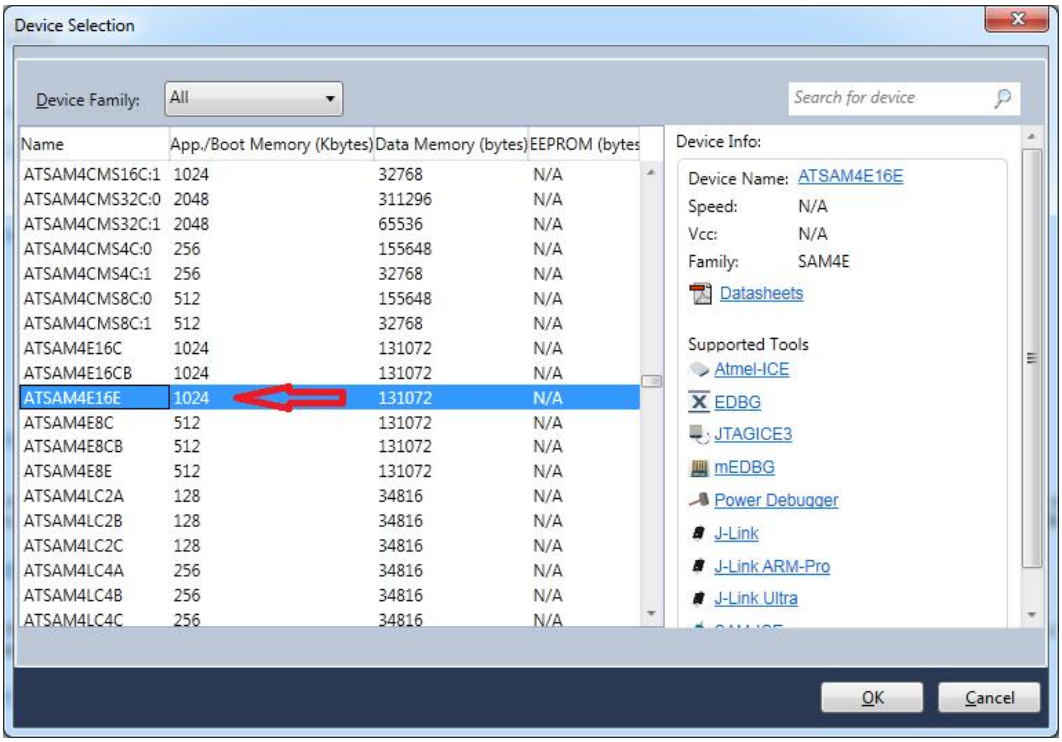


Рис. 4.14. Вибір типу мікросхеми

Написати програму за індивідуальним завданням у вказаному місці:

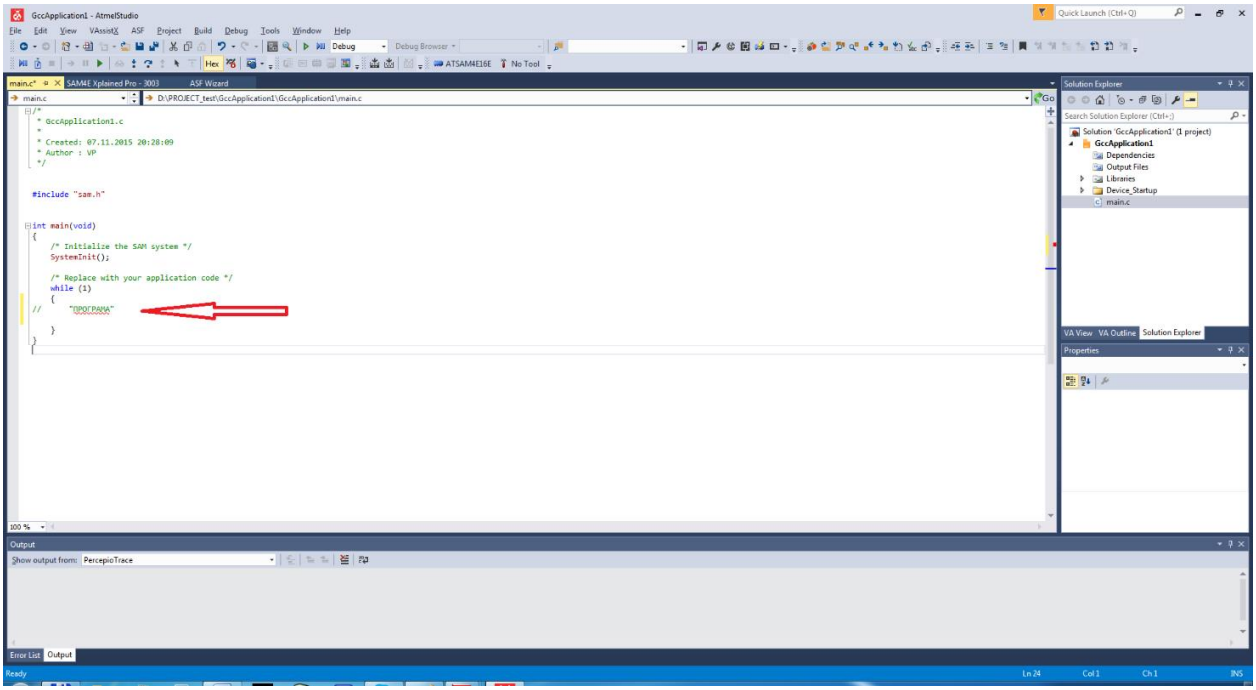


Рис. 4.15. Підготовка програми

Вибір режиму СИМУЛЯТОРА (**Simulator**) відлагодження програми для мікроконтролера **ATSAM4E16E**

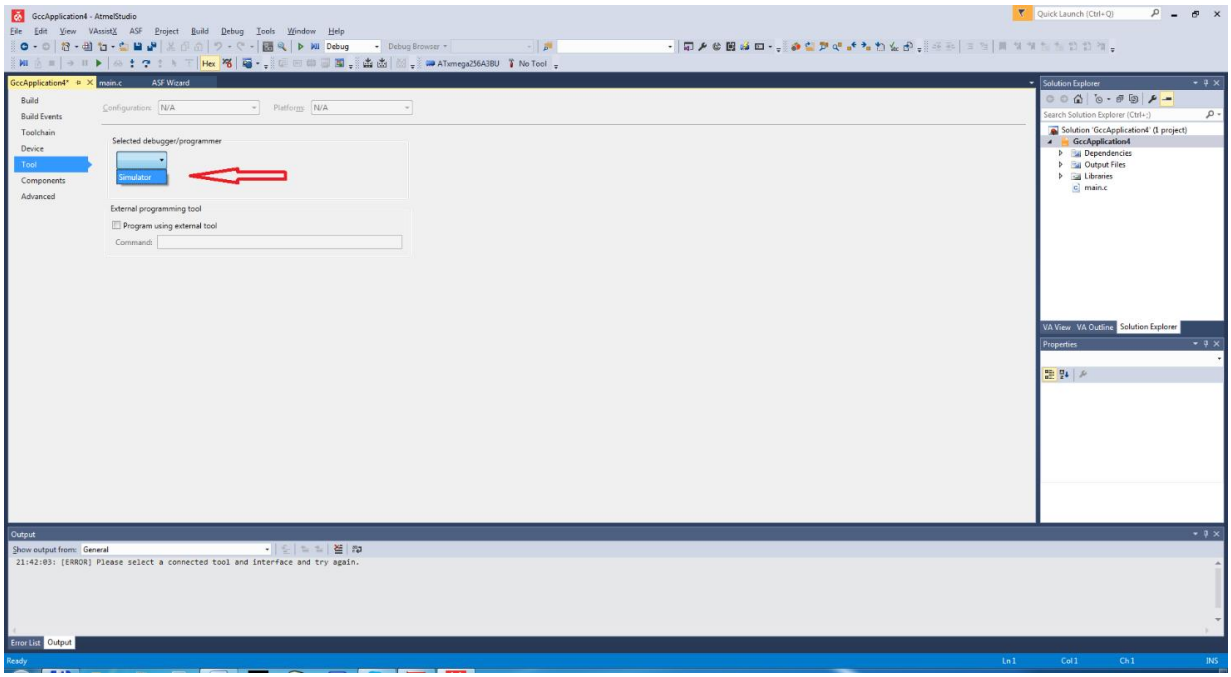


*Рис. 4.16. Вибір режиму Simulator для мікроконтролера **ATSAM4E16E***

Ознайомитися з функціонуванням температурного сенсора DS1621 та функціонуванням двох-провідного інтерфейсу TWO (I2C) в розділі “Методичні матеріали”.

Запустити тестову програму вимірювання температури навколишнього середовища з допомогою температурного сенсора DS1621, в процесі покрокового виконання програми проконтролювати значення температури величиною “t” у вікні Watch 2:

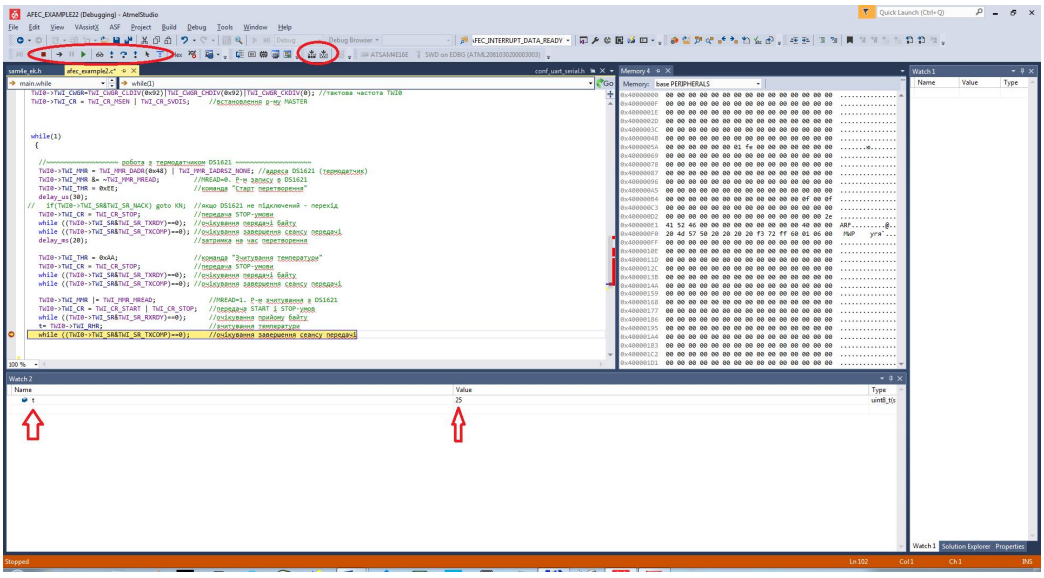


Рис. 4.17. Вікно відлагоджувача

МЕТОДИЧНІ МАТЕРІАЛИ

Основні характеристики ATSAM4E16E

Ядро

ARM® Cortex®-M4 with 2 Kbytes Cache running at up to 120 MHz
Memory Protection Unit (MPU)
DSP Instruction
Floating Point Unit (FPU)
Thumb®-2 Instruction Set

Пам'ять

Up to 1024 Kbytes Embedded Flash
128 Kbytes Embedded SRAM
16 Kbytes ROM with Embedded Boot Loader Routines (UART) and IAP Routines
Static Memory Controller (SMC): SRAM, NOR, NAND Support.
NAND Flash Controller.

Периферійні пристрої

Embedded Voltage Regulator for Single Supply Operation
Power-on-Reset (POR), Brown-out Detector (BOD) and Dual Watchdog for Safe Operation
Quartz or Ceramic Resonator Oscillators: 3 to 20 MHz Main Power with Failure Detection and Optional Lowpower
32.768 kHz for RTC or Device Clock
RTC with Gregorian and Persian Calendar Mode, Waveform Generation in Low-power Modes
RTC Clock Calibration Circuitry for 32.768 kHz Crystal Frequency Compensation
High Precision 4/8/12 MHz Factory Trimmed Internal RC Oscillator with 4 MHz Default Frequency for
Device Startup. In-application Trimming Access for Frequency Adjustment
Slow Clock Internal RC Oscillator as Permanent Low-power Mode Device Clock
One PLL up to 240 MHz for Device Clock and for USB
Temperature Sensor
Up to 2 Peripheral DMA Controller with up to 33 Channels (PDC)
One 4-channel DMA Controller

Two USARTs with USART1 (ISO7816, IrDA®, RS-485, SPI, Manchester and Modem Modes)
USB 2.0 Device: Full Speed (12 Mbits), 2668 byte FIFO, up to 8 Endpoints. On-chip Transceiver
Two 2-wire UARTs
Two Two-wire Interfaces (TWI)
High-speed Multimedia Card Interface (SDIO/SD Card/MMC)

One Master/Slave Serial Peripheral Interface (SPI) with Chip Select Signals

Three 3-Channel 32-bit Timer/Counter with Capture, Waveform, Compare and PWM Mode. Quadrature

Decoder Logic and 2-bit Gray Up/Down Counter for Stepper Motor

32-bit Real-time Timer and RTC with Calendar and Alarm Features

One Ethernet MAC (EMAC) 10/100 Mbps in MII mode only with Dedicated DMA and Support for IEEE1588,

Wake-on-LAN

Two CAN Controllers with eight Mailboxes

4-channel 16-bit PWM with Complementary Output, Fault Input, 12-bit Dead Time Generator Counter for

Motor Control

Real-time Event Management

Cryptography

AES 256-bit Key Algorithm compliant with FIPS Publication 197

AFE (Analog Front End): 2x16-bit ADC, up to 24-channels, Differential Input Mode, Programmable Gain

Stage, Auto Calibration and Automatic Offset Correction

One 2-channel 12-bit 1 Msps DAC

One Analog Comparator with Flexible Input Selection, Selectable Input Hysteresis

I/O

Up to 117 I/O Lines with External Interrupt Capability (Edge or Level Sensitivity), Debouncing, Glitch

Filtering and On-die Series Resistor Termination

Bidirectional Pad, Analog I/O, Programmable Pull-up/Pull-down

Five 32-bit Parallel Input/Output Controllers, Peripheral DMA Assisted Parallel Capture Mode

ВНУТРІШНЯ СТРУКТУРА ATSAM4E16E

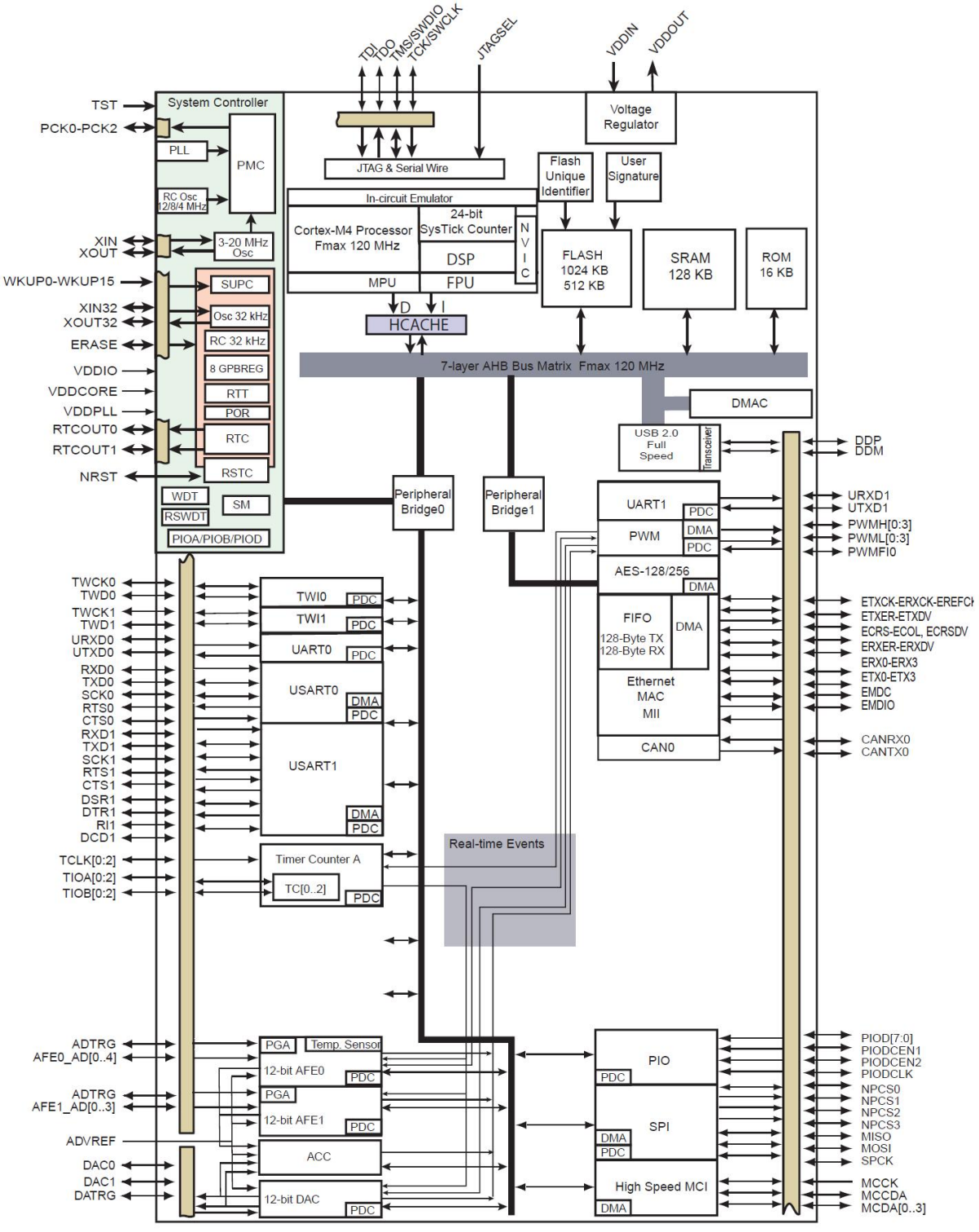


*Рис. 4.18а. Внутрішня структура **ATSAM4E16E***

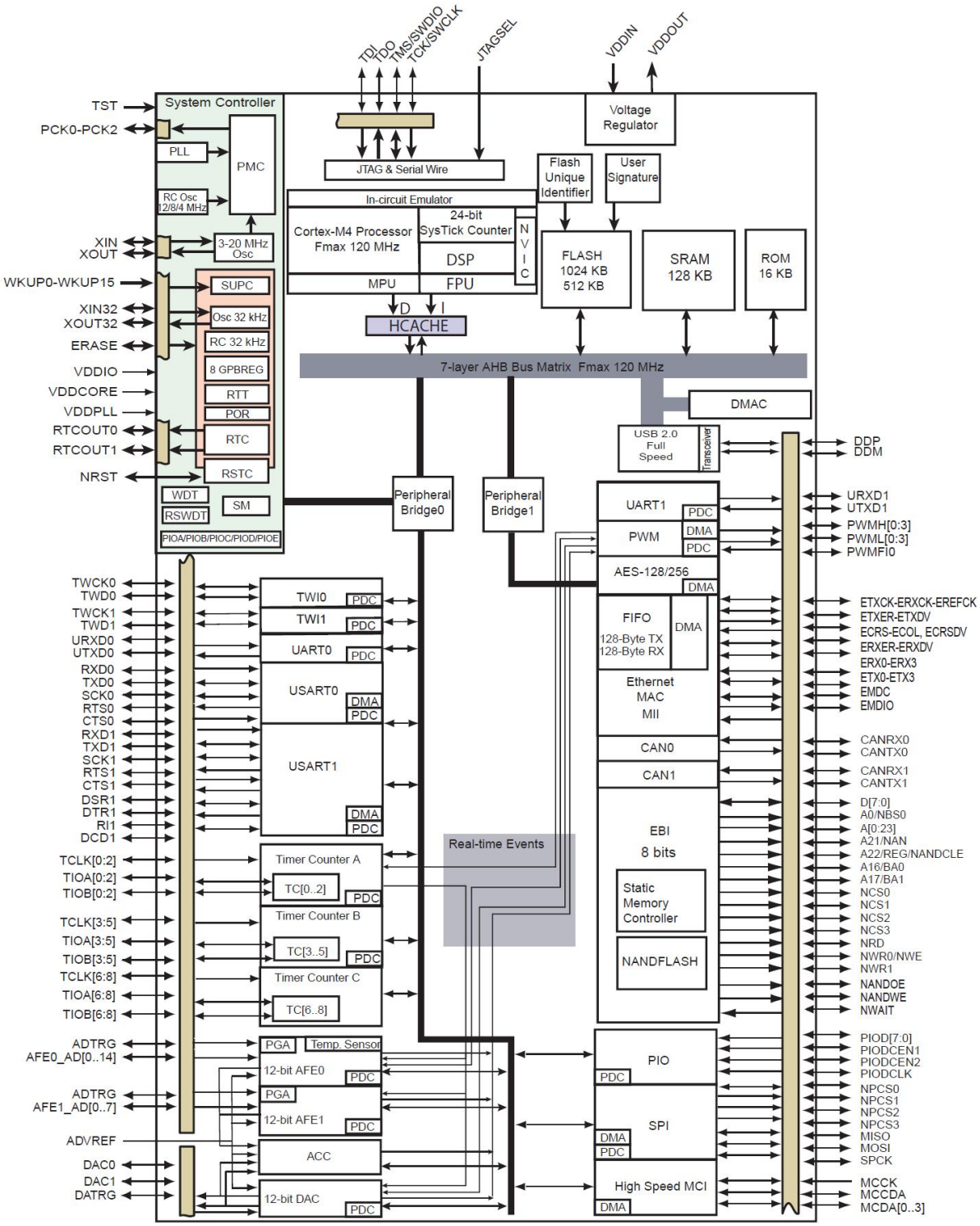


*Рис. 4.18б. Внутрішня структура **ATSAM4E16E***

СТРУКТУРА ПАМ'ЯТІ

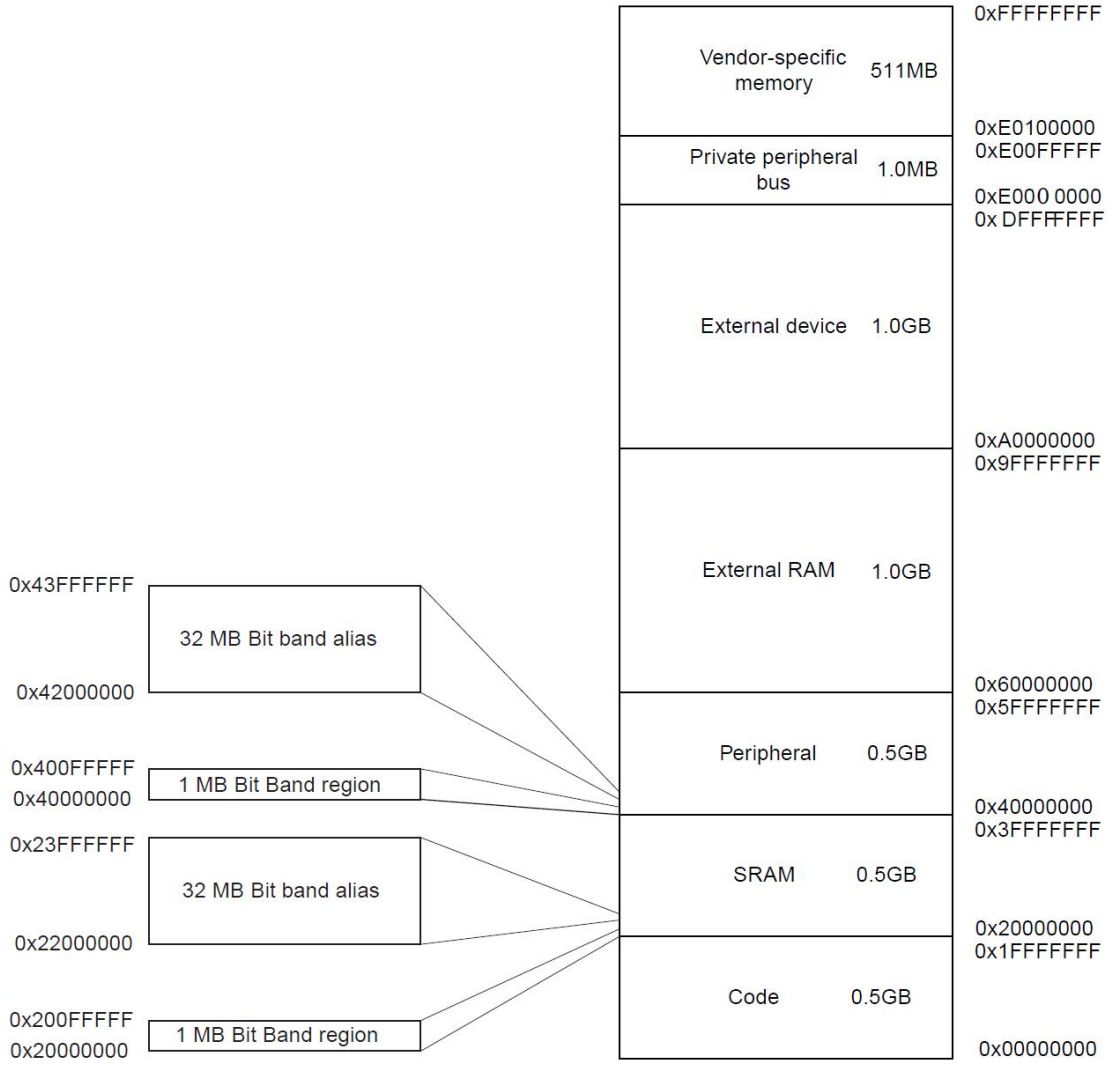


Рис. 4.19. Адресний простір пам'яті

Starter Kit SAM4E-Xplained-Pro

Основні вузли:

ATSAM4E16E microcontroller
- Embedded debugger (EDBG)
- USB interface
- Programming and debugging (target) through Serial Wire Debug (SWD)
- Virtual COM-port interface to target via UART
- Atmel Data Gateway Interface (DGI) to target via USART or TWI
- Four GPIOs connected to target for code instrumentation
- Digital I/O
- Two mechanical buttons (user, reset and force wakeup button)

- One user LED
- Three extension headers
- Xplained Pro LCD extension connector
- One CAN-bus
- 10/100-T Ethernet
- 2Gb 8-bit NAND Flash
- Dual 512K 8-bit SRAM
- Target USB, device mode
- Three possible power sources
- External power
- Embedded debugger USB
- Target USB
- 12MHz crystal
- 32kHz crystal

Зовнішній вигляд

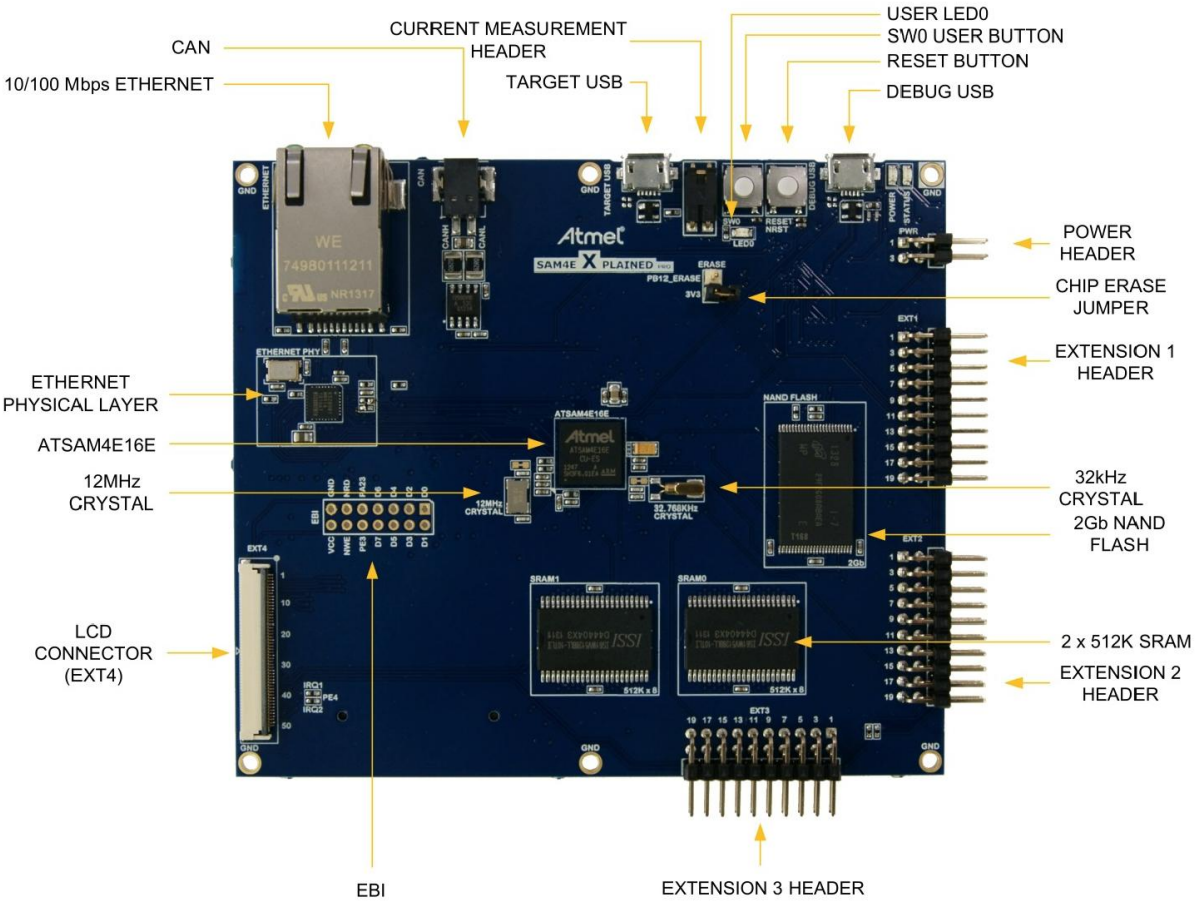


Рис. 4.20. Зовнішній вигляд SAM4E-Xplained-Pro

Схеми відлагоджувального модуля **SAM4E-Xplained-Pro** показані нижче [4.3].

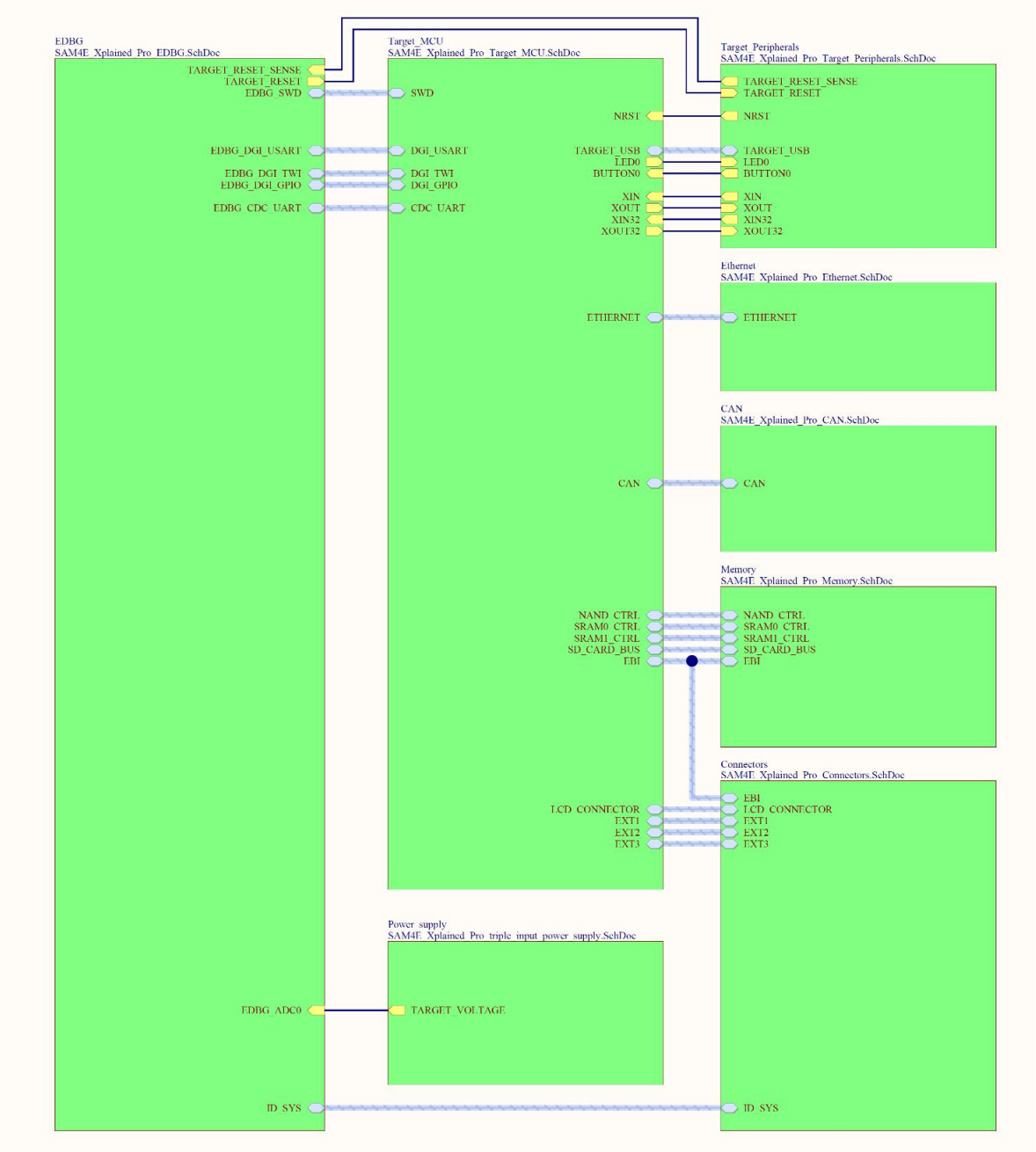


Рис. 4.21. Структура SAM4E-Xplained-Pro

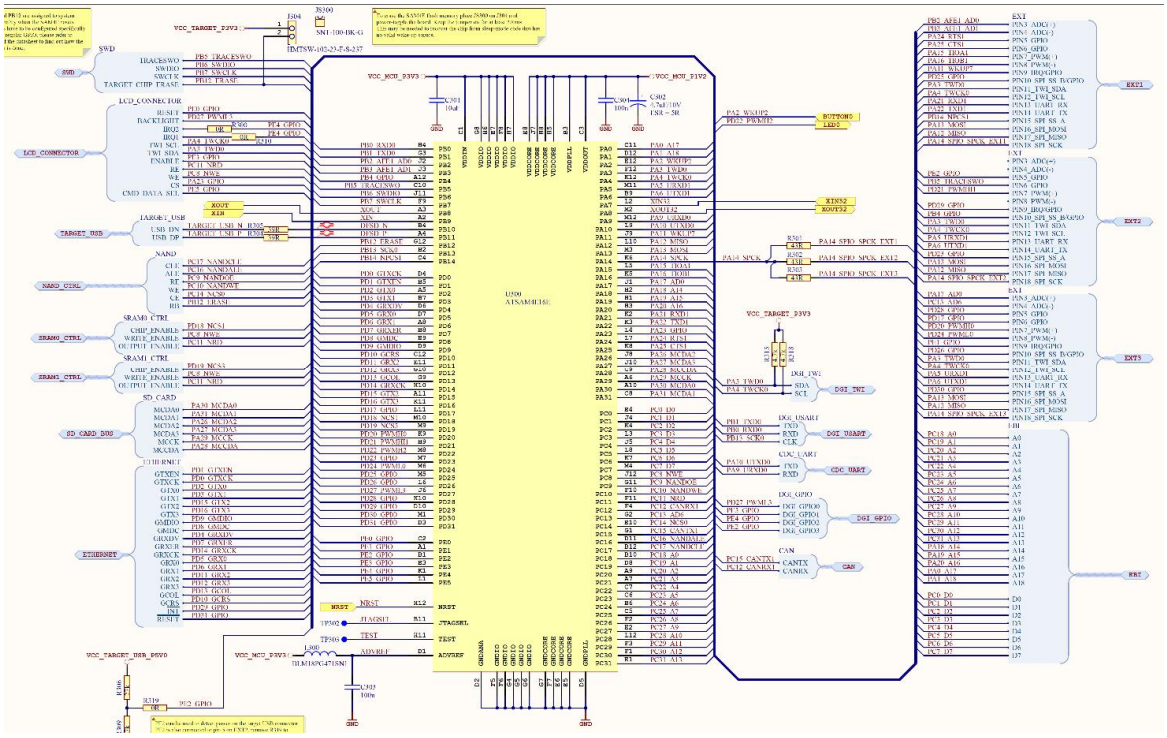


Рис. 4.22. Фрагмент схеми SAM4E-Xplained-Pro

Сенсор температури DS1621

Мікросхема DS1621 це термометр і термостат з цифровим керуванням, гарантує точність вимірювання і контролю з похибкою +- 0,5 гр. С. Керування здійснюється через інтерфейс I2C/TWO.

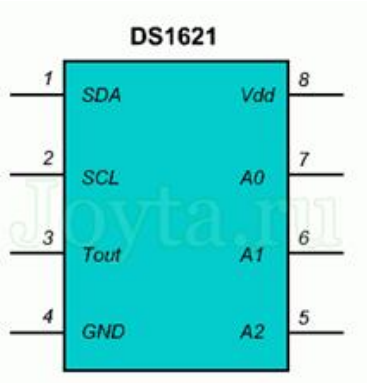


Рис. 4.23. Контакти DS1621

- **SDA –** шина адресу/даних I^2C.
- **SCL –** тактовий сигнал шини I^2C.
- **Tout –** вихід термостата.
- **Vdd –** +5в.
- **GND –** “земля”.
- **A0..A2 –** молодші лінії адресу.

Принцип роботи сенсора DS1621

Принцип вимірювання заснований на нестабільності частоти коливань при зміні температури. Для реалізації цього принципу вимірювання в структуру мікросхеми включені два генератори. Перший з них має високу температурну стабільність. Його робоча частота відповідає температурі – 55 ° С і фактично не змінюється. Робоча частота ж другого генератора, навпаки, змінюється пропорційно зміні температури. Особливі лічильники роблять підрахунок імпульсів за рівний проміжок часу і на базі різниці, проводиться розрахунок поточної температури, який представлений у вигляді 9-разряднго двійкового коду. Дані поділяються на старший і молодший байти. Якщо для будь-яких цілей необхідно ціле значення температури, то потрібно використовувати, лише старший байт. Молодший же байт володіє тільки одним інформаційним бітом – LSB, який реалізує дискретність в 0,5 ° С. Решта біти молодшого байта постійно дорівнюють нулю.

Регістр стану

Мікросхема DS1621 має кілька режимів роботи. Налагодження та контроль даних режимів здійснюється за допомогою “Регістру стану”. До складу регістра входять такі біти:

– **DONE** – біт завершення перетворення. Встановлюється по закінченню перетворення.

– **THF** – біт “висока температура”. Встановлюється при збільшенні температури вище порога TH. Біт скидається програмно або вимиканням живлення.

– **TLF** – біт “низька температура”. Встановлюється при зменшенні температури нижче порога TL. Біт скидається програмно або вимиканням живлення.

– **NVB** – біт запису даних в енергонезалежну пам'ять сенсора. Встановлений біт вказує про те, що запис не завершений. Орієнтовний час зчитування даних складає 10 мс.

– **POL** - вибір полярності виходу Tout. Високе значення відповідає прямій полярності, низький означає зворотну полярність.

– **ISHOT** – біт керування циклом вимірювань. Одноразове вимірювання відбувається при високому логічному рівні даного біта. Його зазвичай застосовують при створенні енергозберігаючих систем. Низький же логічний рівень даного біта, дозволяє виконання перетворення в постійному режимі.

Команди обміну

Обмін даними з датчиком DS1621 відбувається за типовим протоколу I2C. Датчик бере участь в ньому в якості **SLAVE** – пристрої. Його **SLAVE** – адреса має вигляд таку структуру:

1001xxx

де **xxx** – стан ліній **A0-A2** мікросхеми. Для взаємодії з DS1621 застосовуються такі команди:

22h – “Зупинка перетворення” – команда виконує закінчення роботи схеми перетворення температури.

AAh – “Читання температури” – Результат роботи команди – два байта даних, які містять величину виміряної температури.

A1h – “Установка TH” – команда вибору верхнього порогу спрацьовування термостата. Після цієї команди потрібна передача двох байтів значення порогу.

A2h – “Установка TL” – команда вибору нижнього порогу спрацьовування термостата. Після даної команди потрібна передача двох байтів значення порогу.

A8h – “читання температурного лічильника”. Команда діє тільки на читання і дозволяє прочитати дані лічильника, частота роботи якого залежить від температури.

A9h – “читання стабільного лічильника”. Команда діє тільки на читання і дозволяє прочитати дані лічильника, частота роботи якого не залежить від температури.

ACh – “Регістр конфігурації”. Якщо біт дорівнює R – проводиться запис регістра конфігурації, при W – читання.

EEh – “Старт лічильника” – команда запуску вимірювання температур

Two-wire Interface (TWI/I2C)

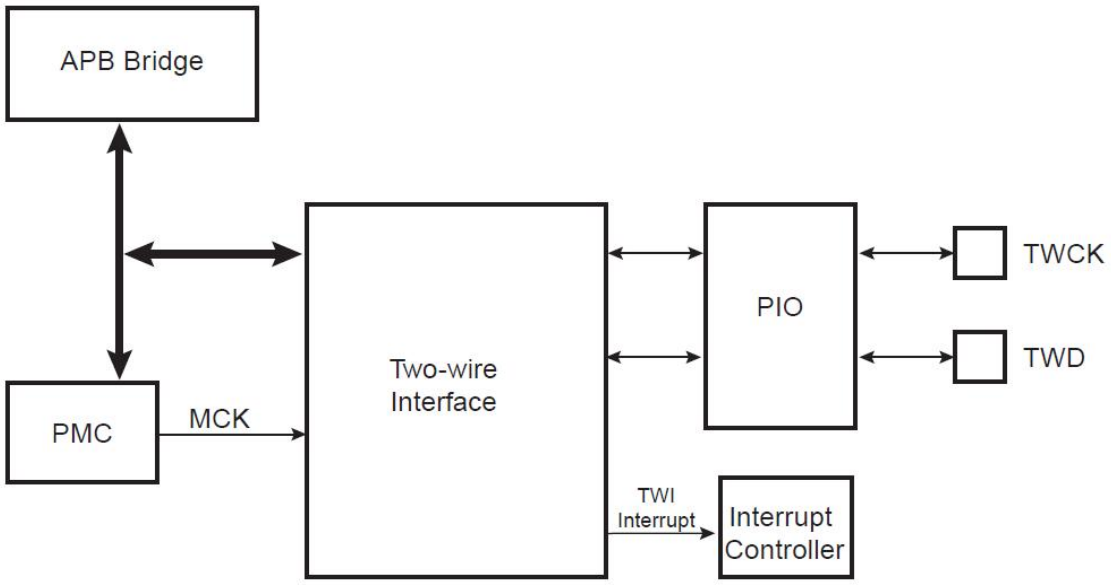


Рис. 4.24. Внутрішня структура

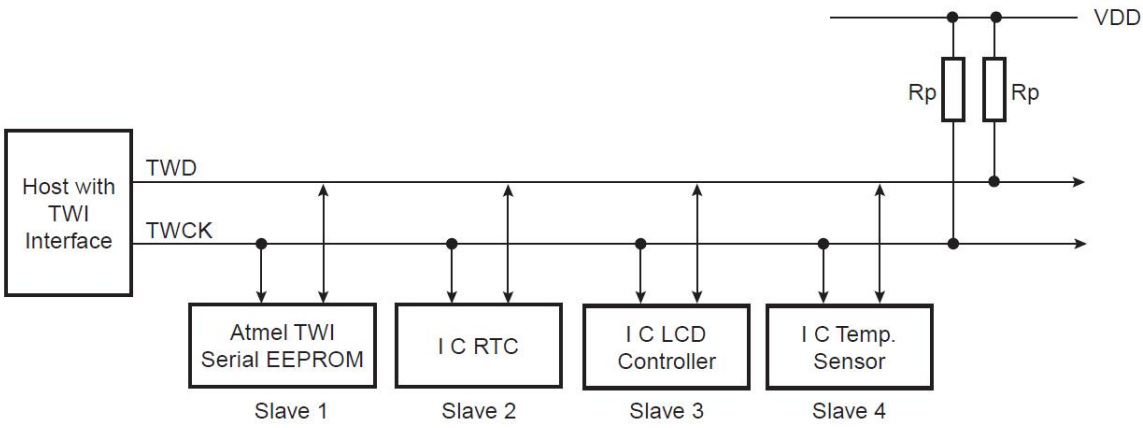


Рис. 4.25. Структура шини

Таблиця 4.1. Сигнали інтерфейсу

Instance	Signal	I/O Line	Peripheral
TWI0	TWCK0	PA4	A
TWI0	TWD0	PA3	A
TWI1	TWCK1	PB5	A
TWI1	TWD1	PB4	A

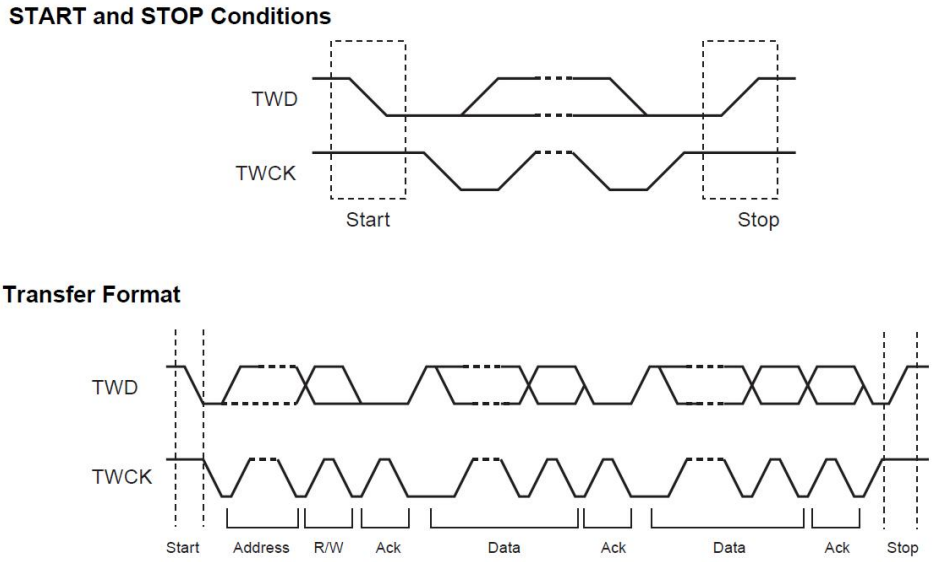


Рис. 4.26. Часові діаграми шини

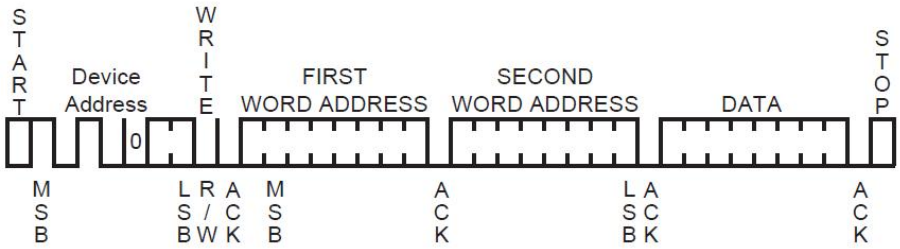


Рис. 4.27. Адресація пристроїв на шині

Література

4.1. https://www.microchip.com/wwwproducts/en/ATSAM4E16E

4.2. https://www.microchip.com/mplab/avr-support/atmel-studio-7

4.3. SAM4E FPU and CMSIS DSP Library. https://www.microchip.com/wwwproducts/en/ATSAM4E16E

4.4. SAM4E-Xplained-Pro_User-Guide.pdf. https://www.microchip.com/wwwproducts/en/ATSAM4E16E

4.5. DS1621.pdf. Maxim Integrated Products, Inc.

4.6. Конспект лекцій з дисципліни “Мікропроцесорні системи”

Лістинг afec_example2.c

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "asf.h"

bool f_1ms;                             //признак 1-ї мілісекунди
uint32_t vidl32;                //черговий відлік
uint8_t t;                              //температура


//----------- brief Configure UART console ----------------
static void configure_console(void)
{
        const usart_serial_options_t uart_serial_options = {
                .baudrate = CONF_UART_BAUDRATE,
                .paritytype = CONF_UART_PARITY
        };

        sysclk_enable_peripheral_clock(CONSOLE_UART_ID);        //Configure console UART
        stdio_serial_init(CONF_UART, &uart_serial_options);
}

//========================= ГОЛОВНА ФУНКЦІЯ ==========================
int main(void)
{
 sysclk_init();         //Initialize the SAM system

 board_init();          //ініціалізація портів

 configure_console();   //ініціалізація UART

 //~~~~~~~~~~~~~~~~~~~~~~~~~~ програмування портів ~~~~~~~~~~~~~~~~~~~~~~~~~~
 sysclk_enable_peripheral_clock(ID_PIOA);               //дозвіл частоти PIOA
 PMC->PMC_PCER0 = PMC_PCER0_PID12; //дозвіл PIOD
 PIOD->PIO_WPMR = 0x50494F;
 PIOD->PIO_OER = PIO_OER_P22;           //PD22 - вихід

 PMC->PMC_PCER0 = PMC_PCER0_PID9;   //дозвіл PIOA
 PIOA->PIO_WPMR = 0x50494F;
 PIOA->PIO_OER = PIO_OER_P21;           //PA21 - вихід
 PIOA->PIO_OER = PIO_OER_P22;           //PA22 - вихід
 PIOA->PIO_OER = PIO_OER_P12;           //PA12 - вихід
 PIOA->PIO_OER = PIO_OER_P13;           //PA13 - вихід
 PIOA->PIO_OER = PIO_OER_P14;           //PA14 - вихід
 PIOA->PIO_OER = PIO_OER_P15;           //PA15 - вихід
 PIOA->PIO_OER = PIO_OER_P24;           //PA24 - вихід
 PIOA->PIO_OER = PIO_OER_P25;           //PA25 - вихід
```

```
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~ програмування TWI0 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 PMC->PMC_PCER0 = PMC_PCER0_PID17;                          //дозвіл TWI0
 sysclk_enable_peripheral_clock(ID_TWI0);            //дозвіл частоти TWI0
 TWI0->TWI_WPMR = 0x50494F;
 PIOA->PIO_IDR = PIO_IDR_P3;                                //переключення PA3,PA4 на
TWI
 PIOA->PIO_IDR = PIO_IDR_P4;
 PIOA->PIO_PDR = PIO_PDR_P3;
 PIOA->PIO_PDR = PIO_PDR_P4;

 TWI0-
>TWI_CWGR=TWI_CWGR_CLDIV(0x92)|TWI_CWGR_CHDIV(0x92)|TWI_CWGR_CKDIV(0);
//тактова частота TWI0
 TWI0->TWI_CR = TWI_CR_MSEN | TWI_CR_SVDIS;                //встановлення режиму
MASTER



while(1)
  {

  //~~~~~~~~~~~~~~~~~~~~ робота з термодатчиком DS1621 ~~~~~~~~~~~~~~~~~~~~
  TWI0->TWI_MMR = TWI_MMR_DADR(0x48) | TWI_MMR_IADRSZ_NONE; //адреса DS1621
(термодатчик)
  TWI0->TWI_MMR &= ~TWI_MMR_MREAD;                          //MREAD=0. Р-м запису в
DS1621
  TWI0->TWI_THR = 0xEE;                                     //команда "Старт перетворення"
  delay_us(30);
//   if(TWI0->TWI_SR&TWI_SR_NACK) goto KN;     //якщо DS1621 не підключений - перехід
  TWI0->TWI_CR = TWI_CR_STOP;                               //передача STOP-умови
  while ((TWI0->TWI_SR&TWI_SR_TXRDY)==0);   //очікування передачі байту
  while ((TWI0->TWI_SR&TWI_SR_TXCOMP)==0); //очікування завершення сеансу передачі
  delay_ms(20);                                             //затримка на час перетворення

  TWI0->TWI_THR = 0xAA;                                     //команда "Зчитування температури"
  TWI0->TWI_CR = TWI_CR_STOP;                               //передача STOP-умови
  while ((TWI0->TWI_SR&TWI_SR_TXRDY)==0);   //очікування передачі байту
  while ((TWI0->TWI_SR&TWI_SR_TXCOMP)==0); //очікування завершення сеансу передачі

  TWI0->TWI_MMR |= TWI_MMR_MREAD;                                 //MREAD=1. Р-м
зчитування з DS1621
  TWI0->TWI_CR = TWI_CR_START | TWI_CR_STOP;       //передача START і STOP-умов
  while ((TWI0->TWI_SR&TWI_SR_RXRDY)==0);          //очікування прийому байту
  t= TWI0->TWI_RHR;                                               //зчитування
температури
  while ((TWI0->TWI_SR&TWI_SR_TXCOMP)==0);         //очікування завершення сеансу
передачі


  }
}
```

Лабораторна робота № 5

ДОСЛІДЖЕННЯ РЕЖИМІВ КЕРУВАННЯ КОМПОНЕНТАМИ МПС З ДОПОМОГОЮ ПАРАЛЕЛЬНОГО ТА ПОСЛІДОВНОГО ІНТЕРФЕЙСІВ

МЕТА РОБОТИ: ознайомлення з технічними характеристиками та архітектурою мікроконтролера STM32F407, паралельними портами мікроконтролера STM32F407, інтерфейсом I2C, Flash FM24CL16, інтерфесом RS-485, керуванням світлодіодами та реле, основними режимами роботи інтегрованої системи mVision5 фірми KEIL, основними режимами роботи інтегрованого середовища CooCox IDE.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з структурою STM32F407, організацією паралельних портів, інтерфейсом I2C, Flash FM24CL16, організацією інтерфесу RS-485, керуванням світлодіодами та реле.
2. Перевірити свою теоретичну підготовку по контрольних питаннях для самоперевірки; при необхідності скористатися методичними матеріалами.
3. Освоїти основні етапи процесу створення проєкту в системі mVision5 у відповідності до методичних матеріалів.
4. Освоїти основні етапи процесу створення проєкту в середовищі CooCox IDE у відповідності до методичних матеріалів.
5. Завантажити тестову програму до лабораторної роботи № 5 в середовищі CooCox IDE у відповідності до методичних матеріалів.
6. Запустити та налаштувати програму ModbusFS 2.2.
7. Запустити необхідний тестовий проєкт в режимі покрокового виконання.
8. Захистити лабораторну роботу.

ЗАВДАННЯ

1. В покроковому режимі продемонструвати ініціалізацію мікроконтролера та відлагоджувальної плати.
2. В покроковому режимі продемонструвати ініціалізацію паралельних портів.
3. В покроковому режимі продемонструвати ініціалізацію I2C.
4. В покроковому режимі продемонструвати ініціалізацію RS-485 (UART).
5. В покроковому режимі продемонструвати роботу драйвера світлодіодів.
6. В покроковому режимі продемонструвати роботу драйвера реле.
7. В покроковому режимі продемонструвати роботу драйвера RS-485.
8. В покроковому режимі продемонструвати роботу драйвера FM24CL16.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Внутрішня структура мікроконтролера STM32F407, організація паралельних портів, організація інтерфесу RS-485, інтерфейсу I2C, Flash FM24CL16, керування світлодіодами та реле.
2. Програмна модель STM32F407.
3. Основні опції меню відлагоджувача mVision5 та CooCox IDE.

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

Створення проєкту та налагодження програми в інтегрованій системі mVision5

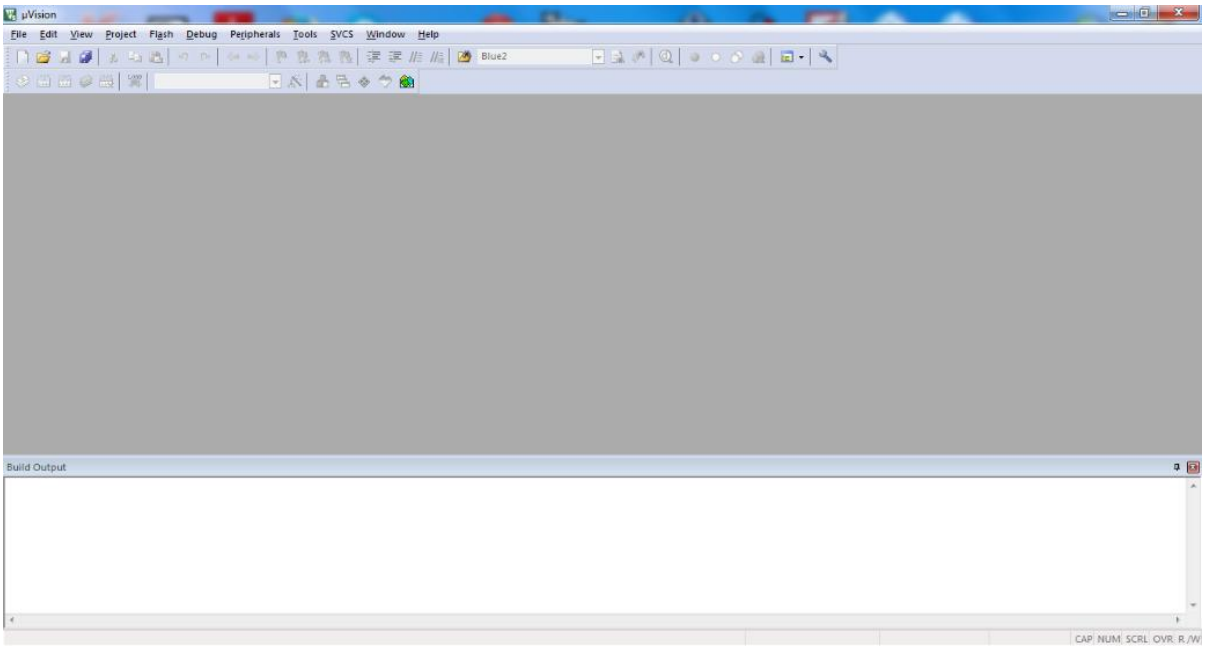


Рис. 5.1. Стартове вікно mVision5

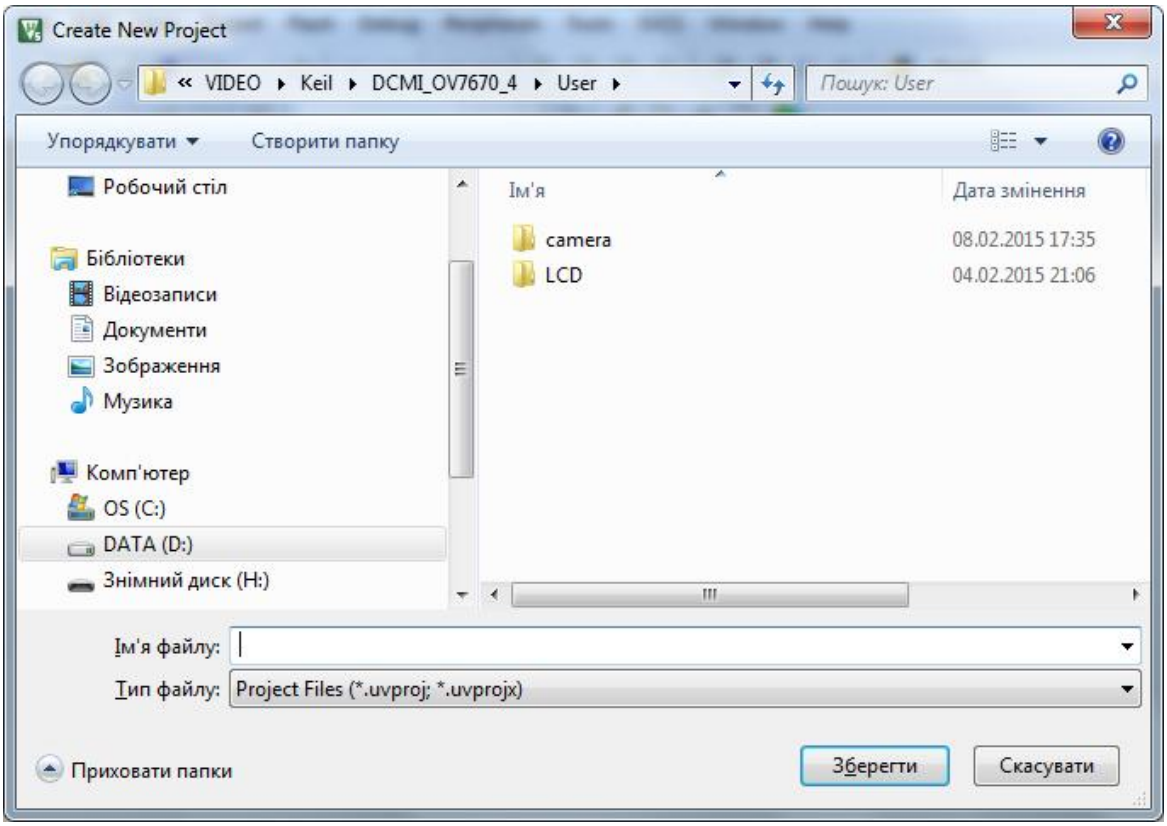


Рис. 5.2. Створення нового проєкту

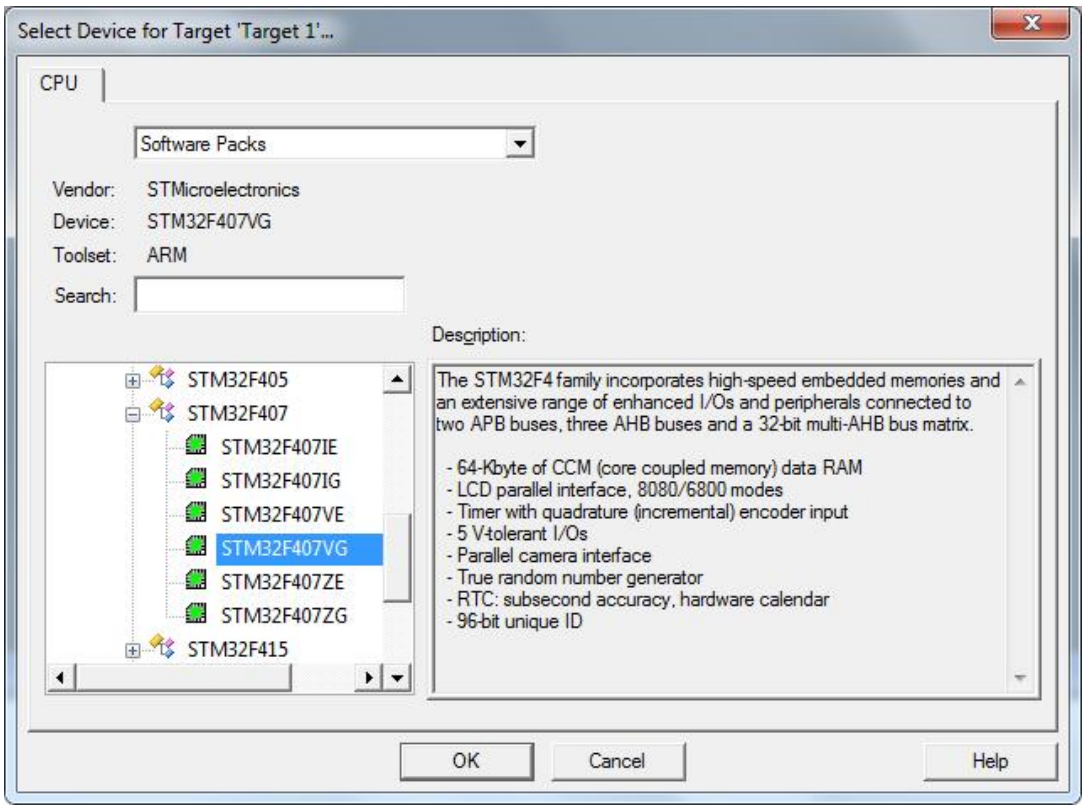


Рис. 5.3. Вікно вибору мікропроцесора STM32F407

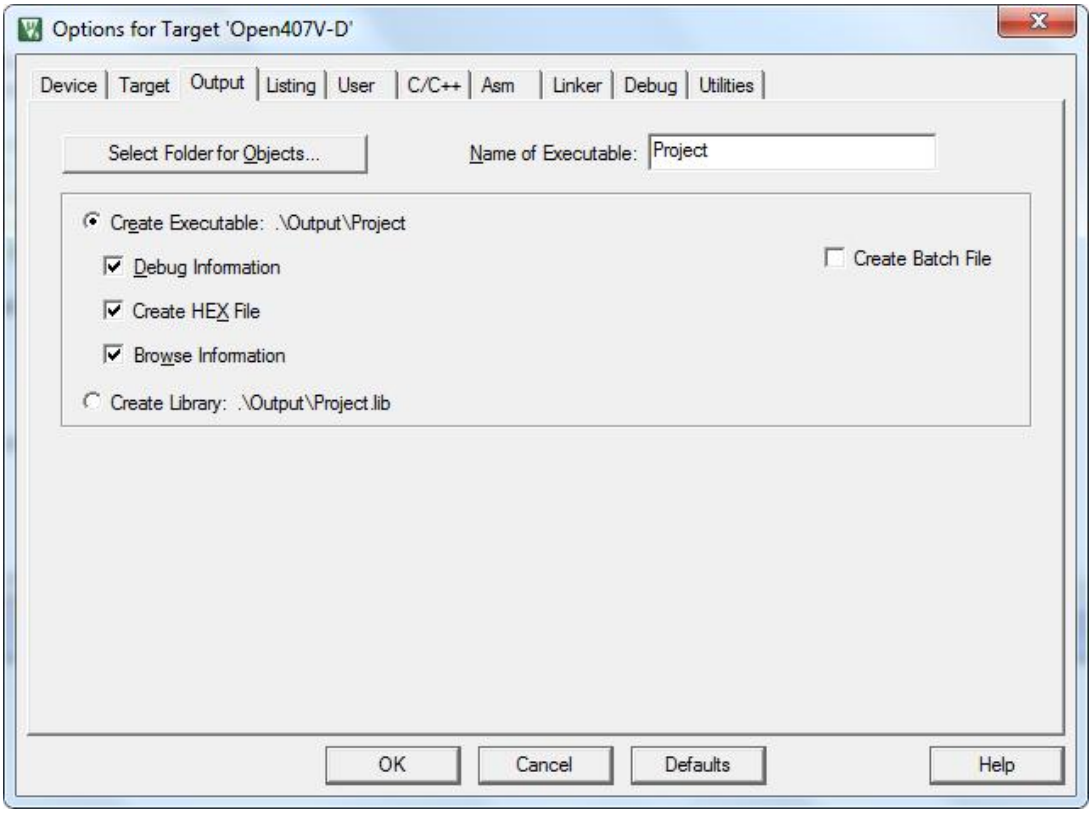


Рис. 5.4. Вікно вибору для сворення HEX-модуля

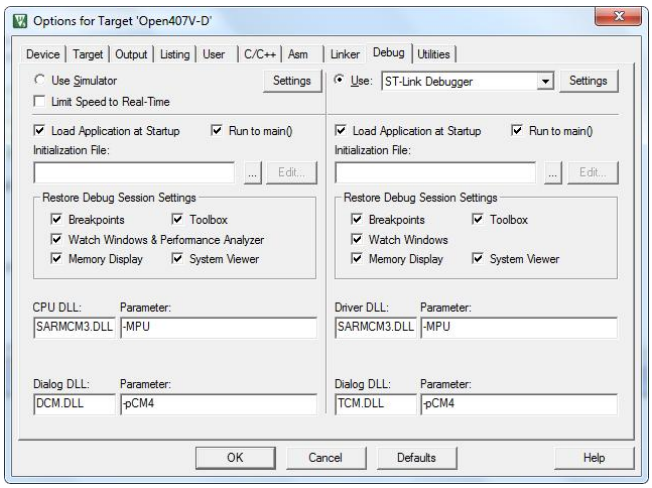


Рис. 5.5. Вікно програми відлагоджувача проєктів

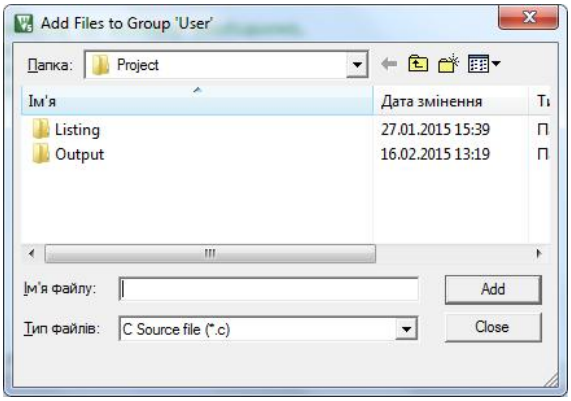


Рис. 5.6. Вікно завантаження програми в проєкт

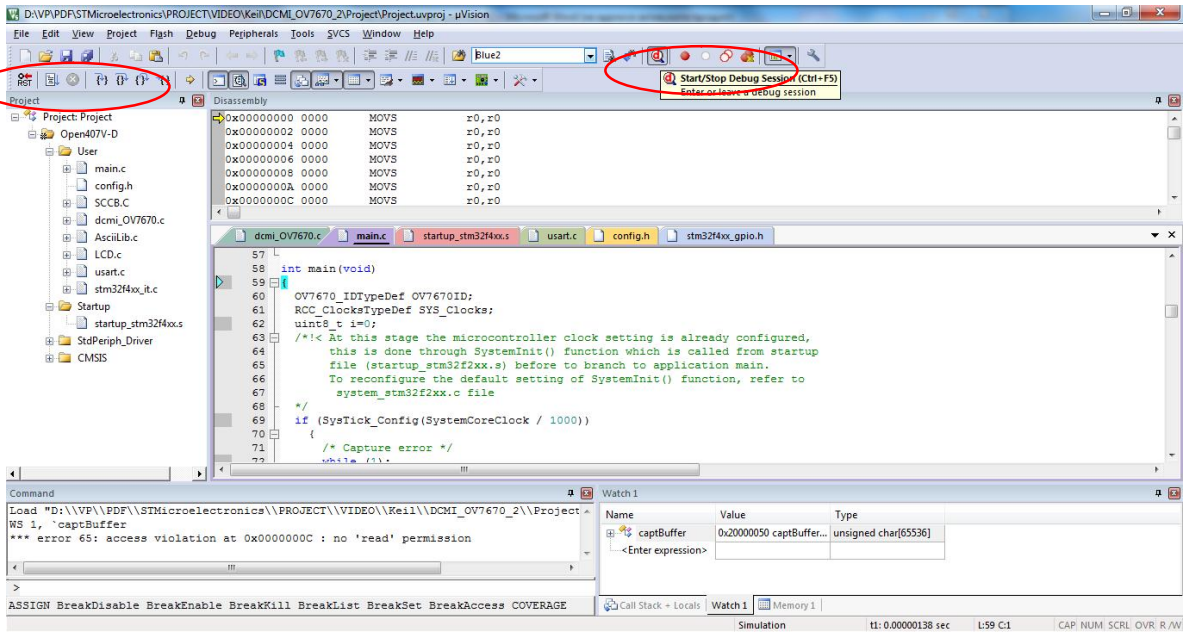


Рис. 5.7. Вікно програми відлагоджувача проєктів, програма завантажена та відкомпільована

Створення проєкту з використанням Pack Installer

Запускаємо Keil 5 та **Pack Installer.**

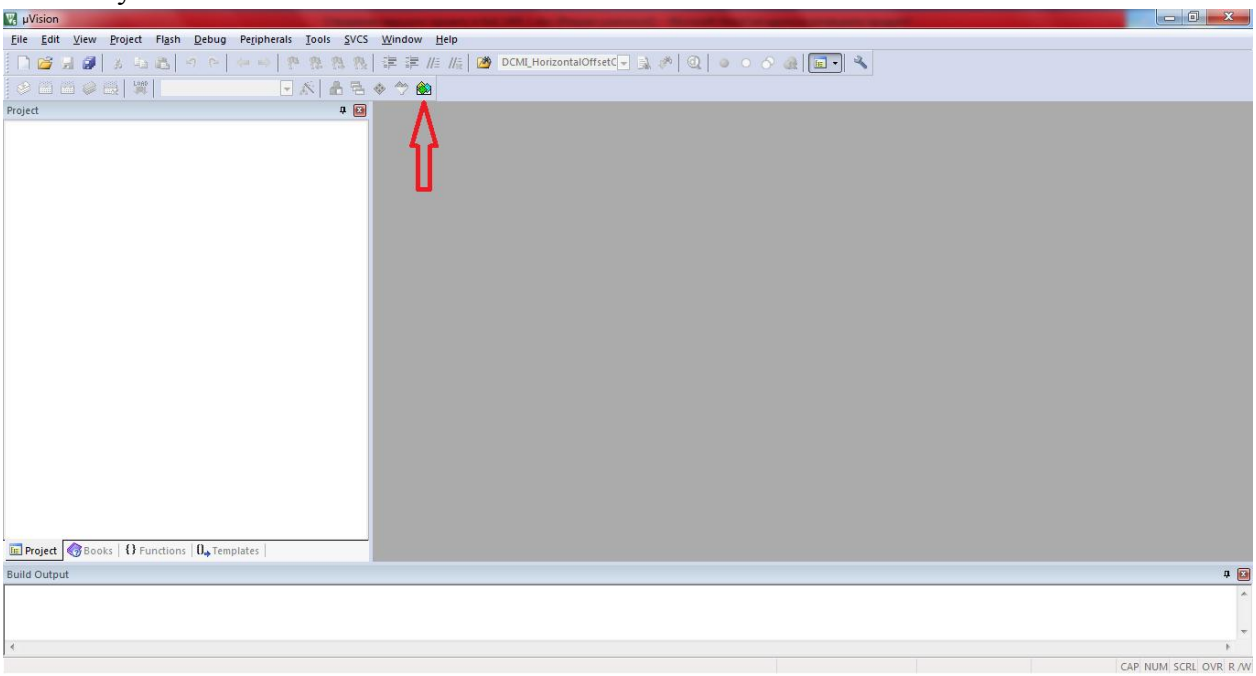


Рис. 5.8. Запуск Pack Installer

На панелі інструментів вгорі натискаємо зазначену кнопку. Відкривається вікно **Pack Installer** в ньому дві вкладки зліва і дві справа

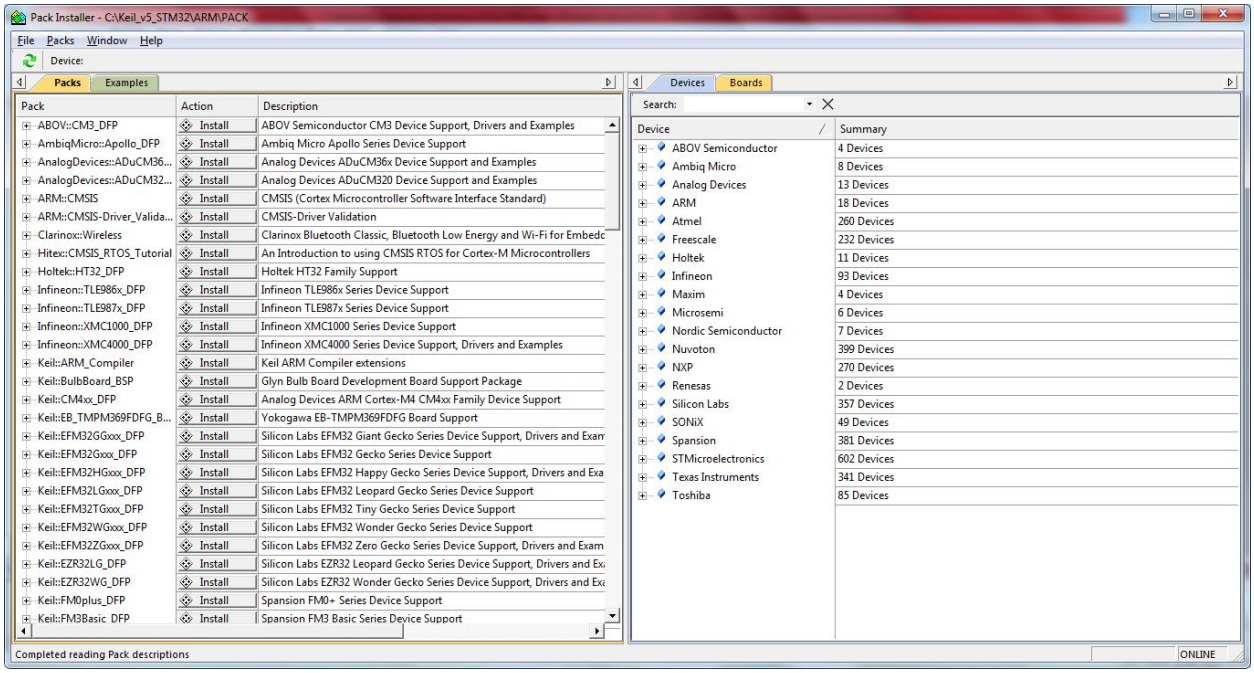


Рис. 5.9. Вибір сімейства мікроконтролера

Натискаємо на вкладку **Devices** вибираємо зі списку виробників STMicroelectronics далі вибираємо серію МК (МК- мікроконтролер) STM та тип МК STM32F407VG. І на вкладці **Pack** по черзі інсталюємо всі пакети.

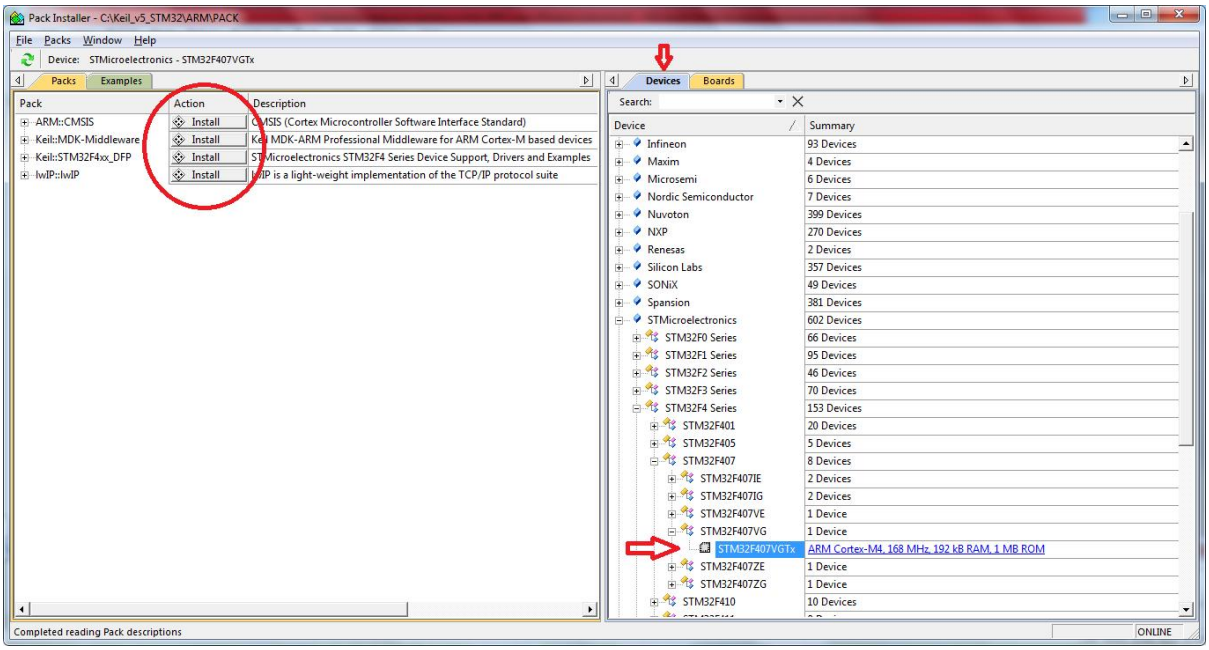


Рис. 5.10. Вибір типу мікроконтролера та інсталяція модулів

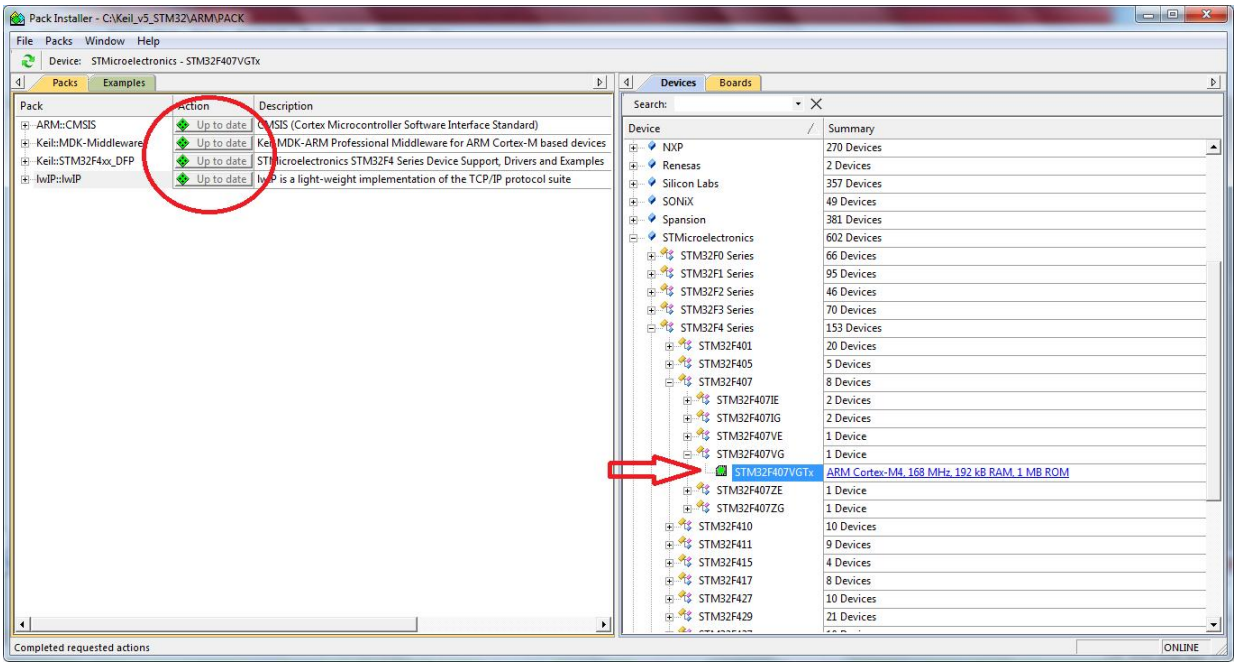


Рис. 5.11. Результат інсталяції модулів

Далі створюємо проєкт: натискаємо кнопку **Project** і у випадаючому меню натискаємо **NewProject**

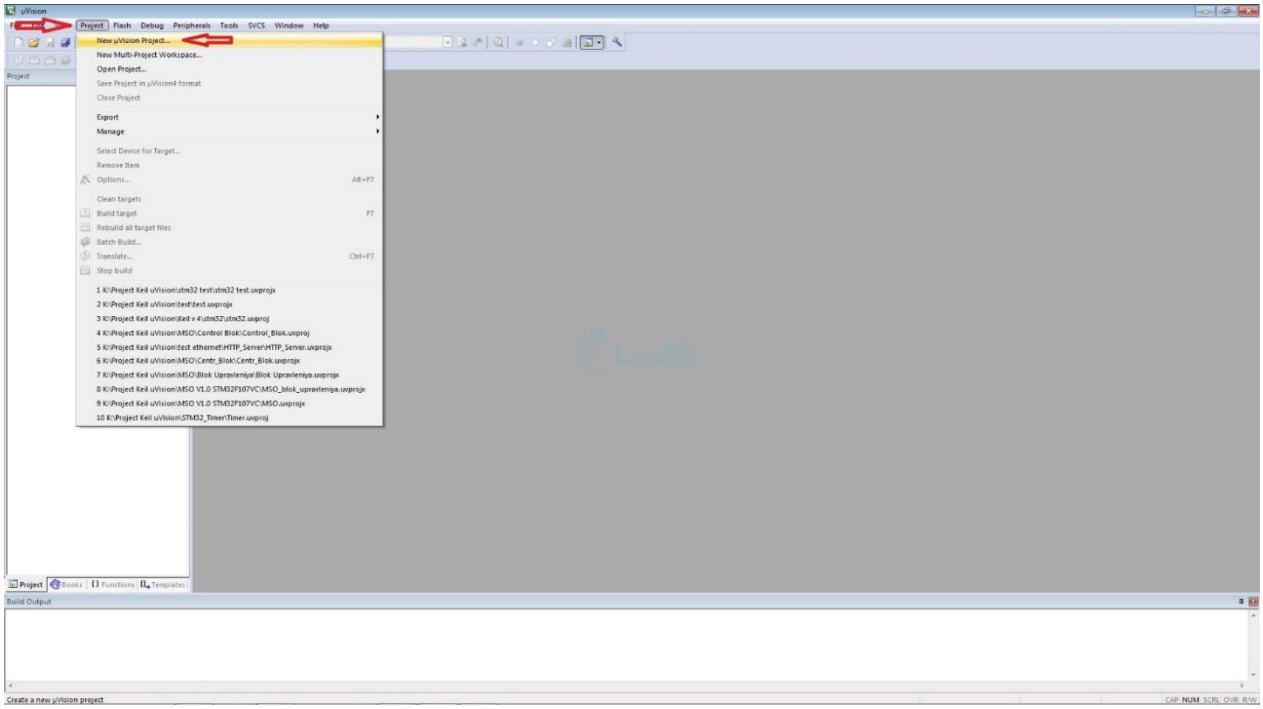


Рис. 5.12. Створення проєкту

З'являється вікно і пропонує вибрати де буде зберігатися проєкт. Далі з'явиться вікно і пропонує вибрати мікроконтролер.

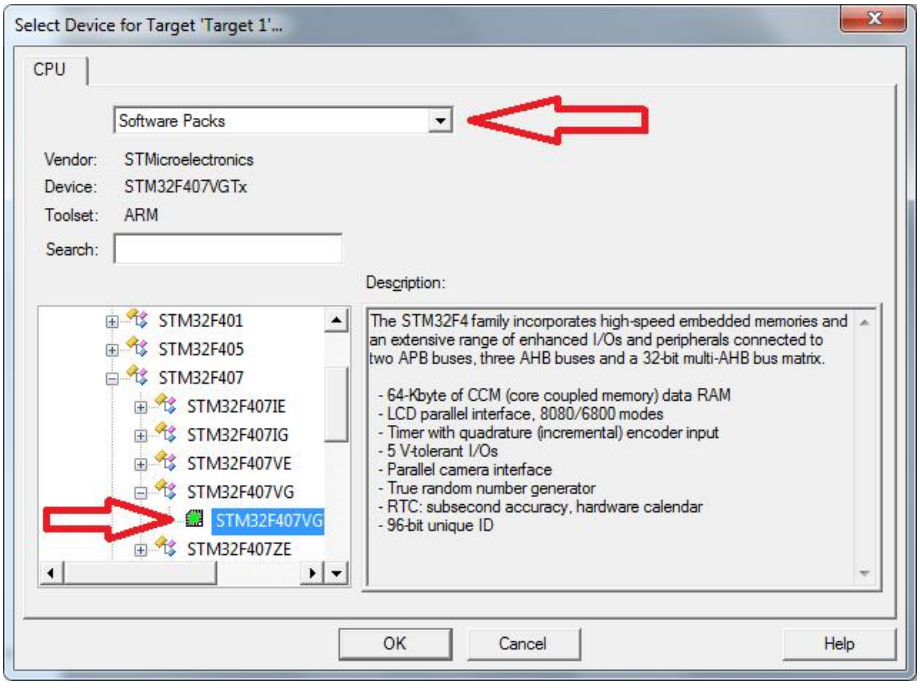


Рис. 5.13. Вибір типу мікроконтролера

Наступне вікно **Manage Components.** Ставимо галочки згідно рисунків:

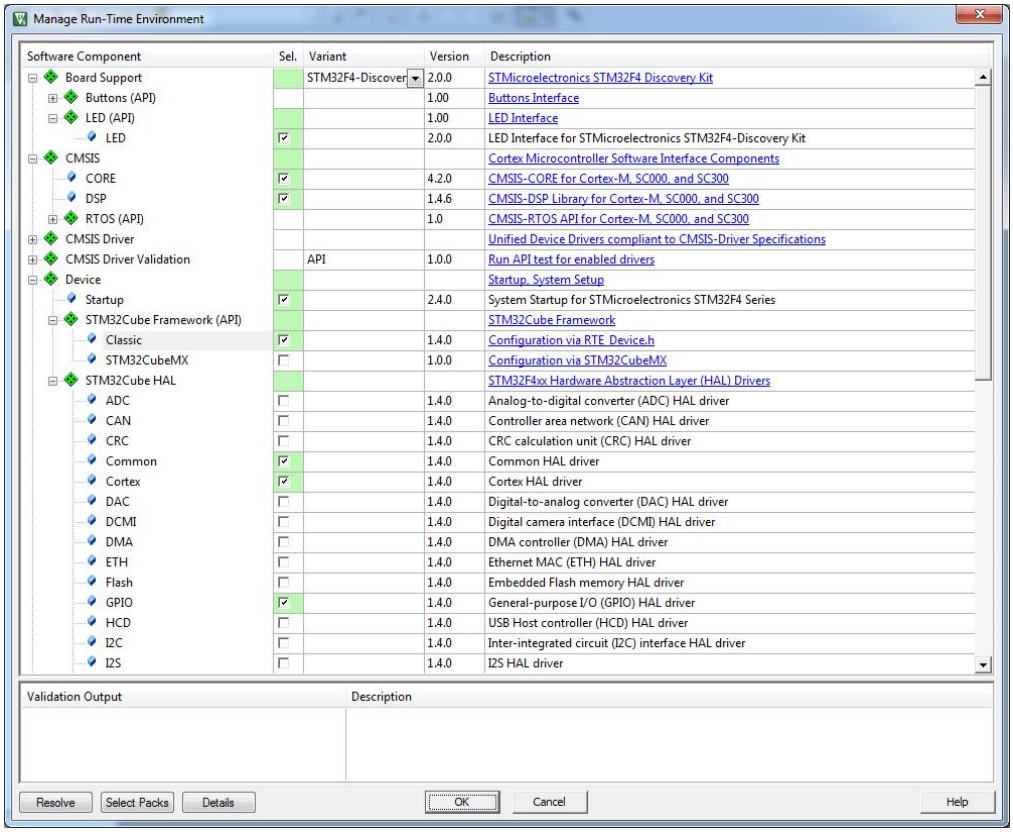


Рис. 5.14а. Вибір модулів

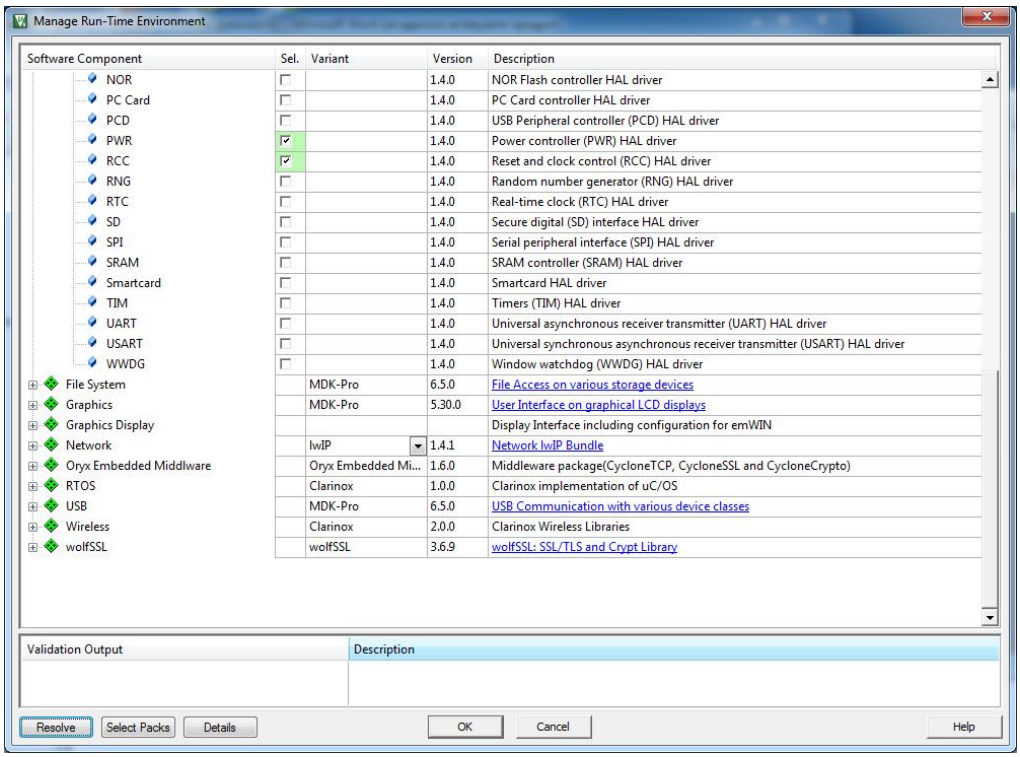


Рис. 5.14б. Вибір модулів

Рядок "**Board Support**" відкриває список підтримуваних налагоджувальних плат, якщо там є ваша то можете вибрати її і поставити галочку Keil завантажить для неї драйвер.

"**CMSIS**" – тут ставимо галочку "**CORE**" це підтримка основного ядра ARM.

"**RTOS API**" – це операційна система реального часу.

CMSIS DRIVER – це драйвера інтерфейсів.

DEVICE – тут міститься практично вся основна периферія мікроконтролера.

Ставимо галочки **GPIO** – це основний драйвер портів введення/виводу,

Startup – це основний конфігураційний системний файл.

Ставимо галочки і натискаємо **ОК**. Якщо у вас квадратики світяться жовтим кольором – натискаєте в самому низу вікна кнопку "**Resolve**" і якщо відповідні драйвери інстальовано то вони стануть зеленими, інакше необхідно доінсталювати відповідні драйвери.

В меню **DEVICE** вибираєте **GPIO** і **Startup**.

Далі в **DEVICE** ставите галочки на тій периферії яка вам потрібна і натискаєте в самому низу вікна кнопку "**Resolve**" потім **ОК** Вікно повинне зникнути. Дальше треба створити файл для коду. Як показано на рисунку правою кнопкою миші по **Source_group** 1 далі **Add new item.**. Далі вибираємо тип файлу. І пишемо його ім'я в полі "**Name**", наприклад, **main**, і натискаємо **ADD**.

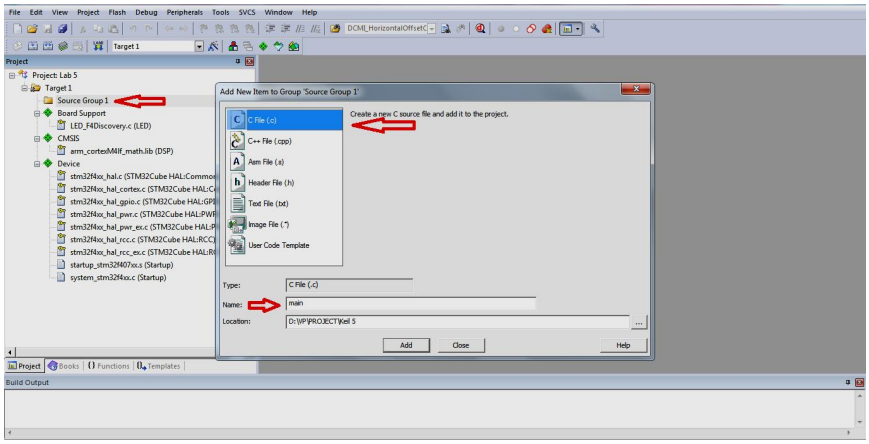


Рис. 5.15. Вибір типу файла

Натискаємо правою кнопкою миші в полі де повинен бути код і вибираємо Insert #include file далі **stm32f4xx.h**. Якщо у вас інший мікроконтролер виконуєте все те ж саме тільки замість **stm32f4xx.h** обираєте свій файл.

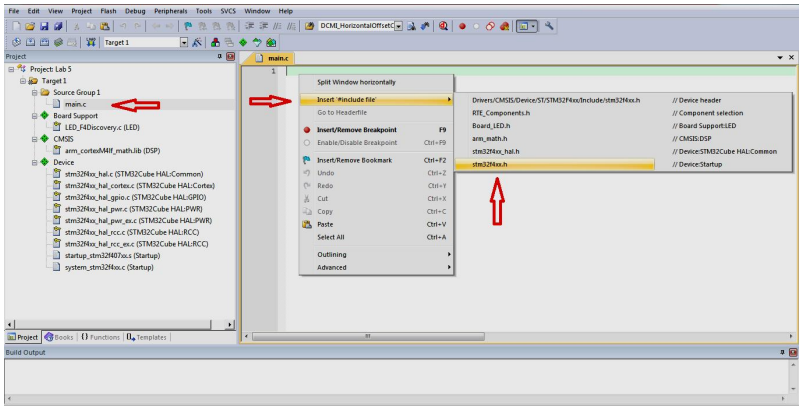


*Рис. 5.16. Підключення файлів типу *.h*

Далі на панелі інструментів натискаємо кнопку "Options for Target"

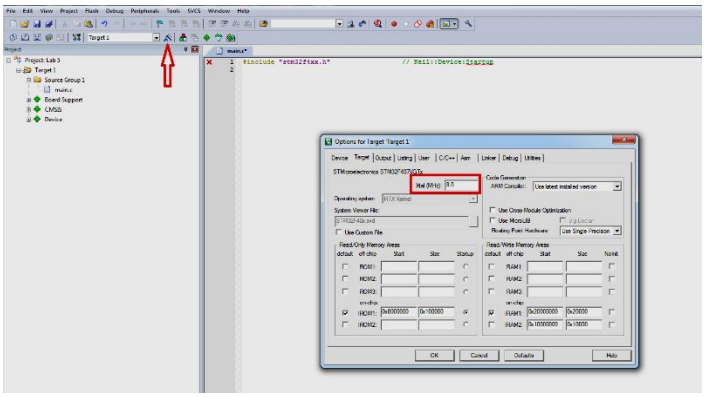


Рис. 5.17. Вибір частоти тактового генератора

Виставляємо частоту Xtal. В наступній вкладці вибрати формування вихідного файлу *.hex : “Create HEX file”

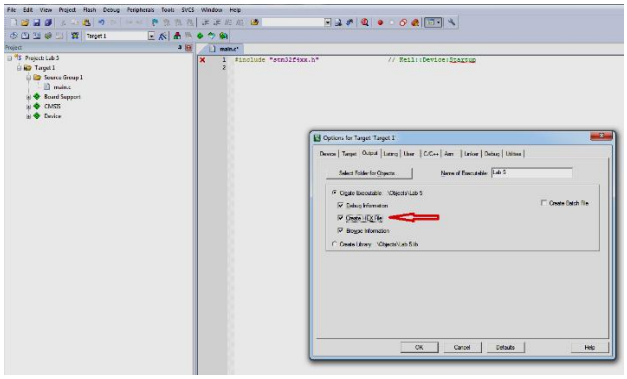


*Рис. 5.18. Вибір формування вихідного файлу *.hex*

Відкриваємо “main” і пишемо програму.

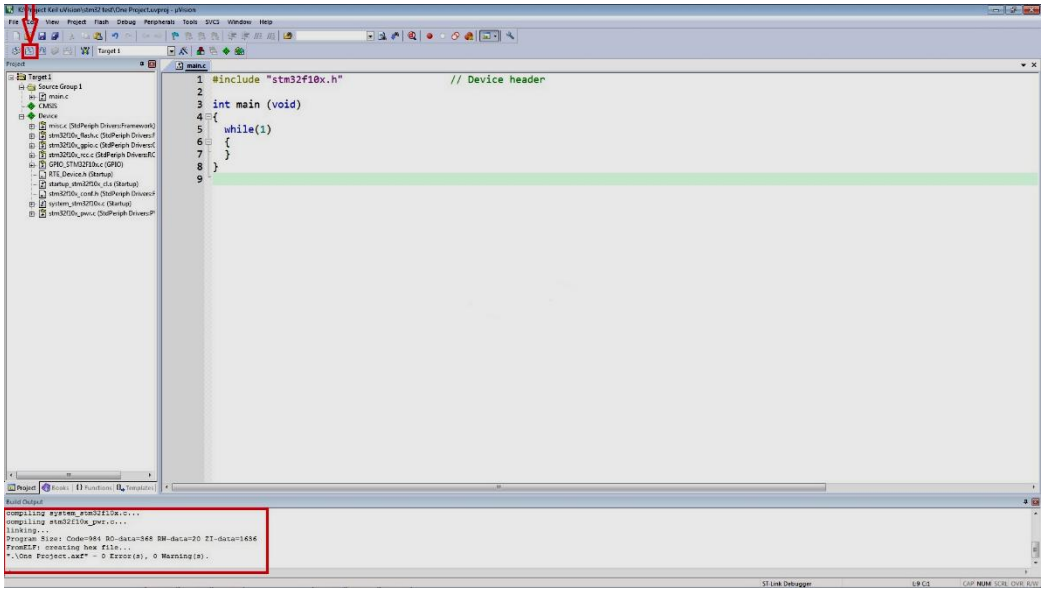


Рис. 5.19. Підготовка та компіляція програми “main”

На панелі інструментів натискаємо кнопку “Build ”.

Відлагодження програми в середовищі Keil 5

Відлагоджувати програму можна з використанням симулятора (“Use Simulator”) або апаратно-програмного відлагоджувача, наприклад, “ST-Link Debugger”, див. рис. 22.

Для відлагодження в реальному часі з використанням апаратної платформи заданого мікроконтролера Keil uVision5 забезпечує підримку широкого набору адаптерів. Крім цього в середовищі є вбудований програматор, який забезпечує програмування пам’яті програм та даних, див. рис. 24.

Модуль STM32F4DISCOVERY. Цей модуль спроєктовано для відлагодження апаратних вузлів мікропроцесорної ситеми на базі мікроконтролерів STM32F407/417. На платі модуля розміщено мікроконтролер STM32F407VGT6 в корпусі LQFP100 з вузлом синхронізації на базі кварцевого резонатора 8MHz, вузол USB, аудіо-ЦАП, 4 світлодіоди користувача, кнопка RESET, кнопка користувача та контактні групи підключені до виводів мікроконтролера. Крім цього на платі розміщено адаптер ST-LINK/V2 який використовується для відлагодження в реальному часі та програмування пам’яті Flash мікроконтролера. Адаптер може бути відключений від мікроконтролера на платі та використовуватися для відлагодження та програмування плати користувача через інтерфейс SWD.

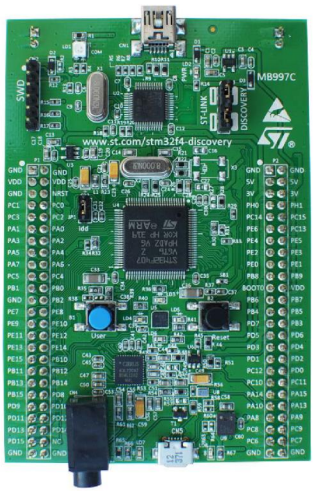


Рис. 5.20. Зовнішній вигляд модуля STM32F4DISCOVERY

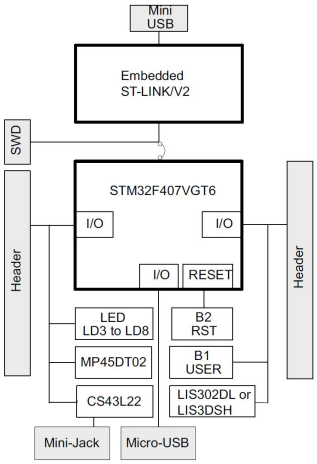


Рис. 5.21. Структура модуля STM32F4DISCOVERY

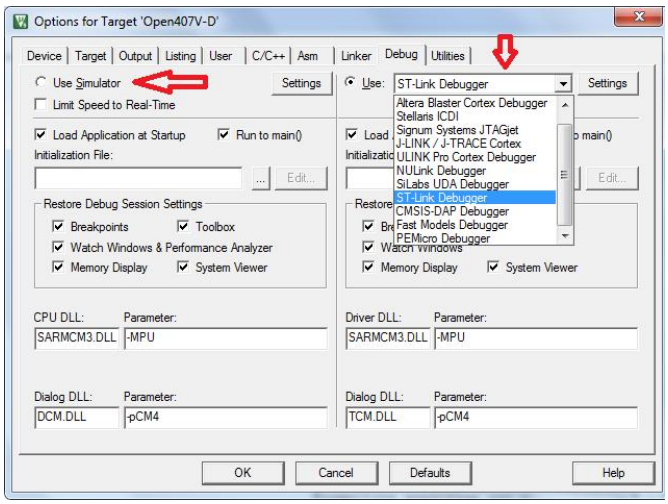


Рис. 5.22. Вікно вибору адаптера для програмування та відлагодження в Keil uVision5

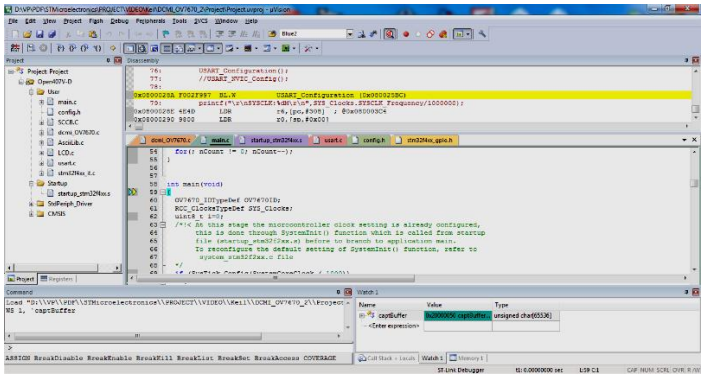


Рис. 5.23. Вікно відлагодження програми в Keil uVision5

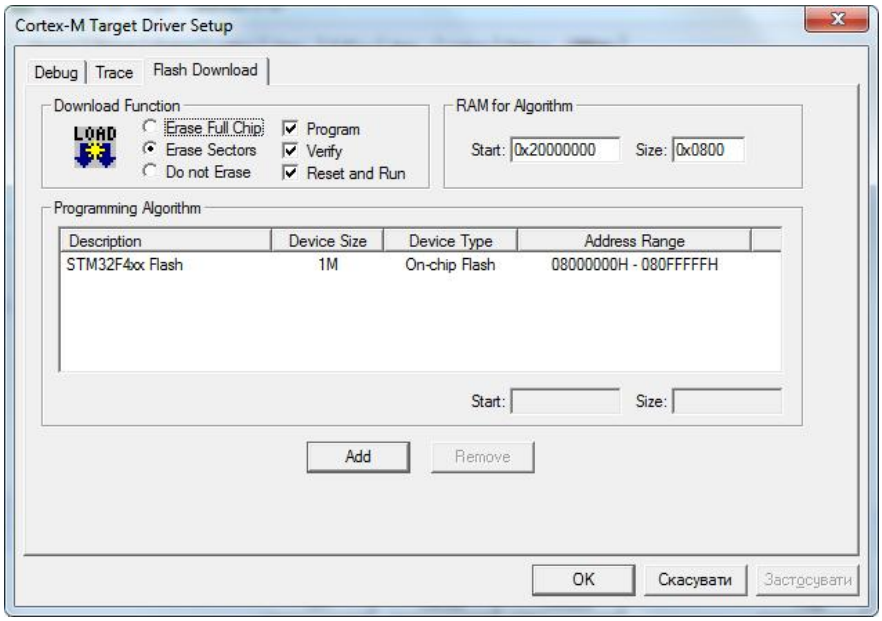


Рис. 5.24. Вікно конфігураці вбудованого програматора в Keil uVision5

Створення проєкту та відлагодження програми в інтегрованій системі CooCox IDE

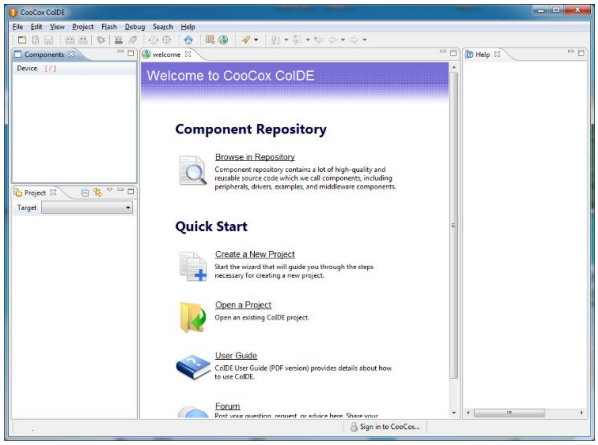


Рис. 5.25. Стартове вікно CooCox IDE

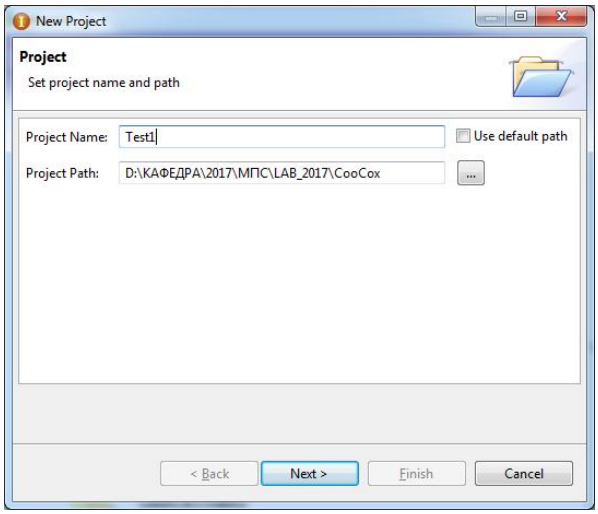


Рис. 5.26. Створення проєкту

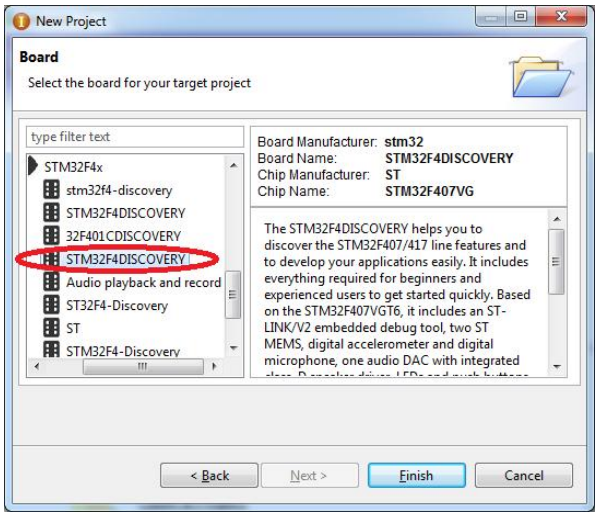


Рис. 5.27. Вибір типу відлагоджувального модуля STM32F407DISCOVERY

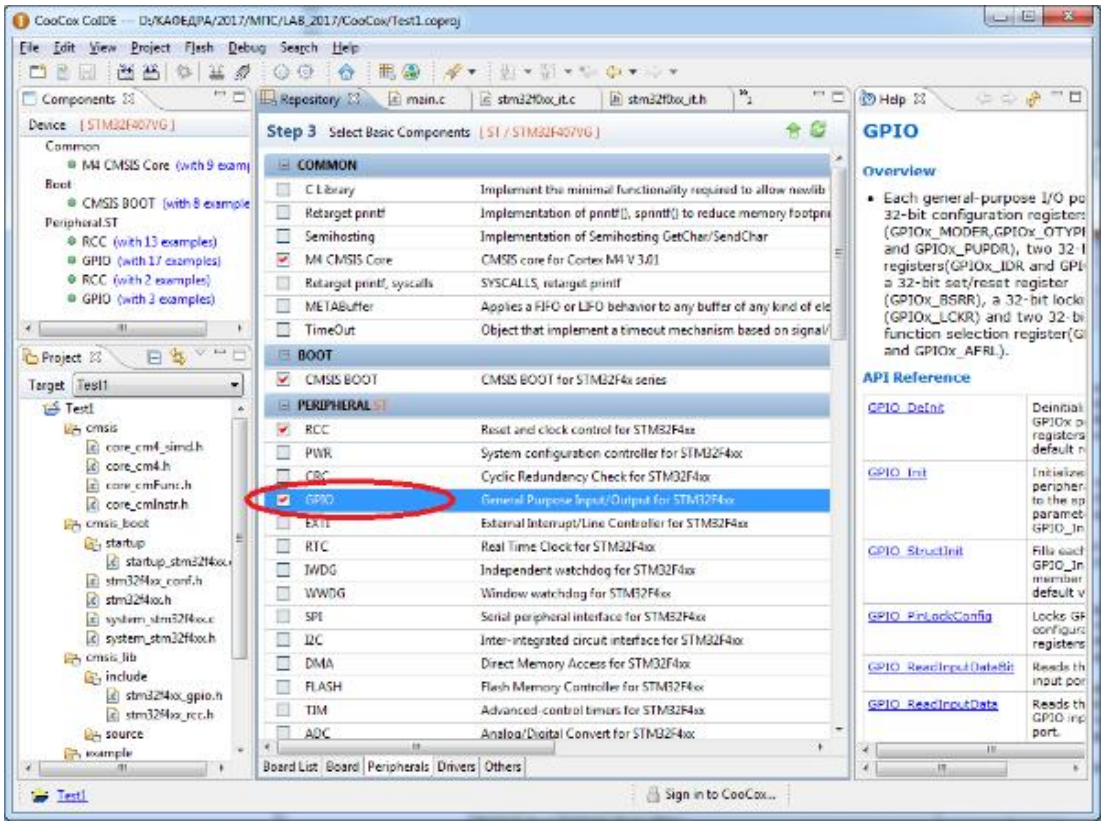


Рис. 5.28. Вибір тестового проєкту для модуля STM32F407DISCOVERY

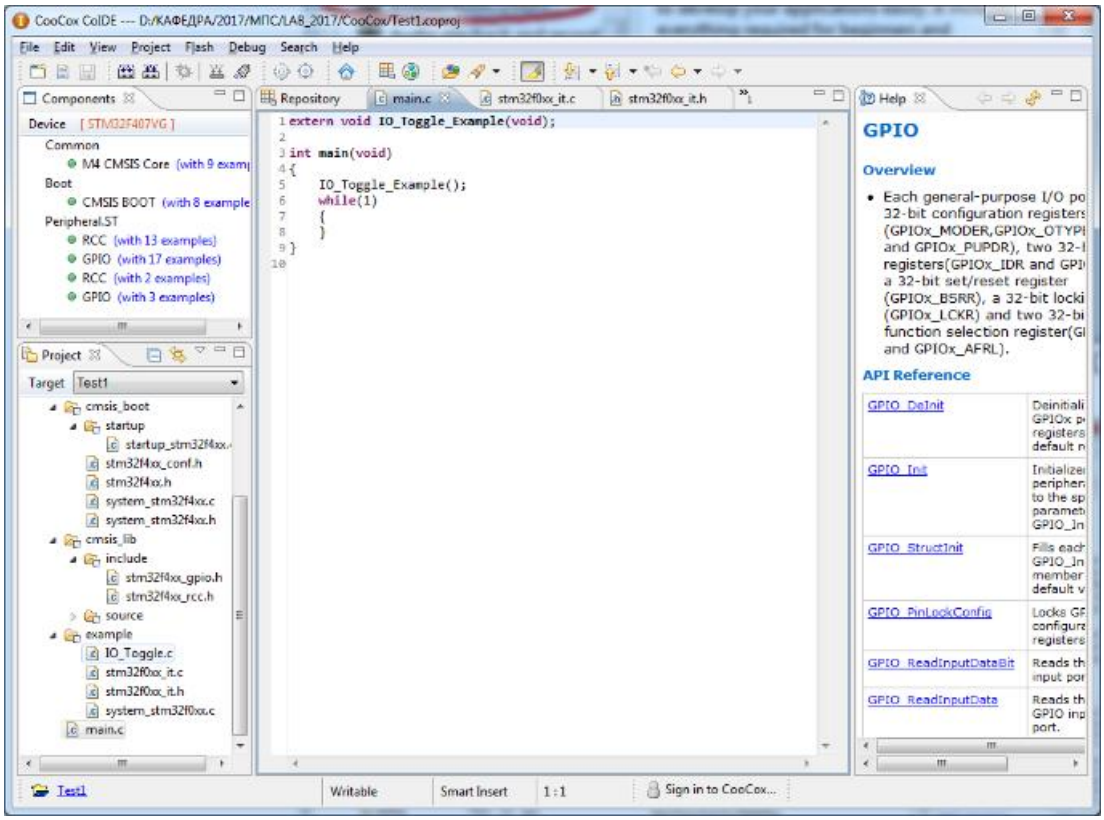


Рис. 5.29. Вікно модуля "main"

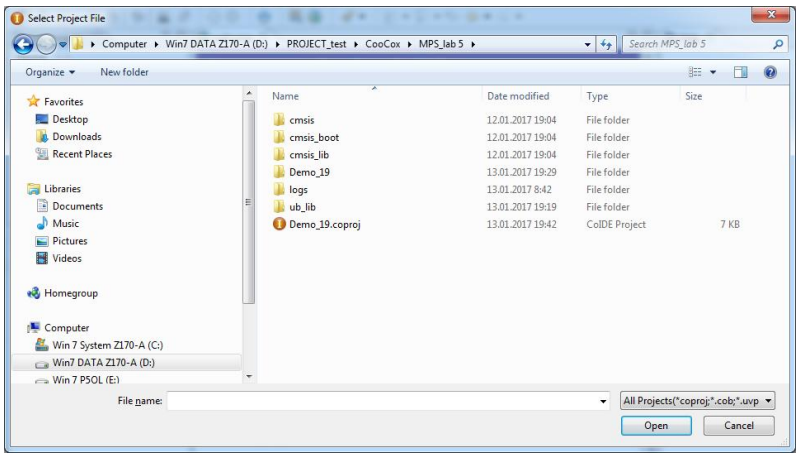


Рис. 5.30. Завантаження готового тестового проєкту в CooCox

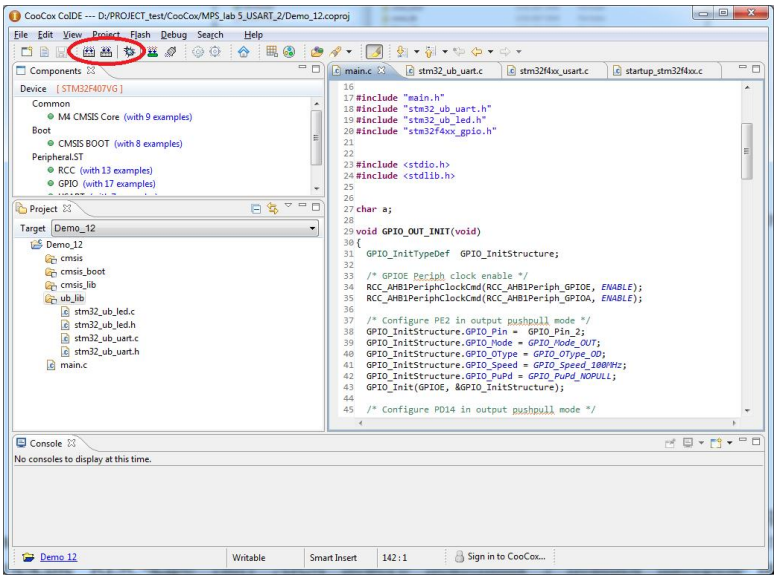


Рис. 5.31. Компіляція та запуск режиму відлагодження тестового проєкту в CooCox

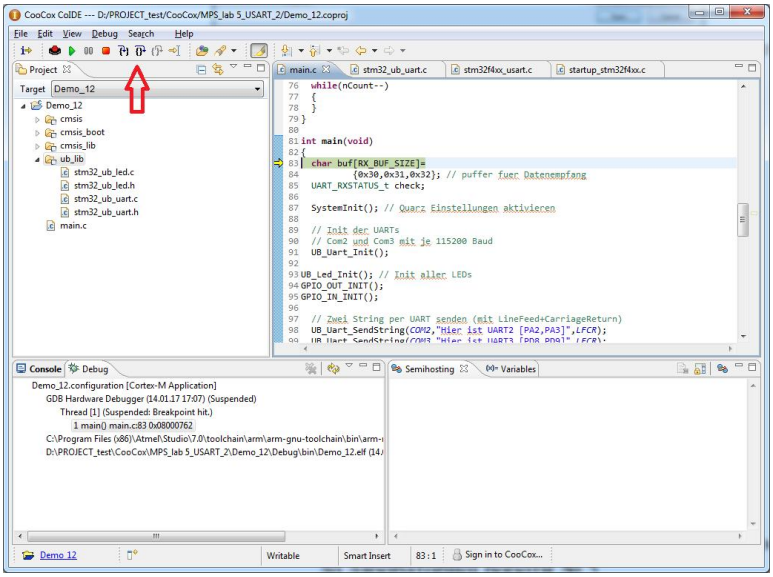


Рис. 5.32. Покрокове виконання тестового проєкту CooCox керування світлодіодами, реле та передачі даних по RS-485

Прийом інформації по RS-485 в ПК здійснюється програмою “Modbus FS”.

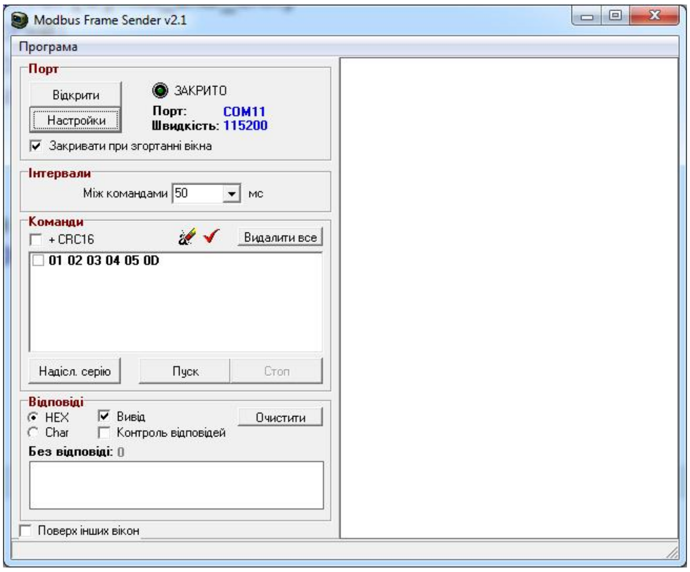


Рис. 5.33. Стартове вікно програми “Modbus FS”

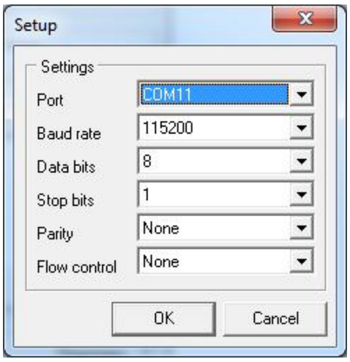


Рис. 5.34. Налаштування програми “Modbus FS”

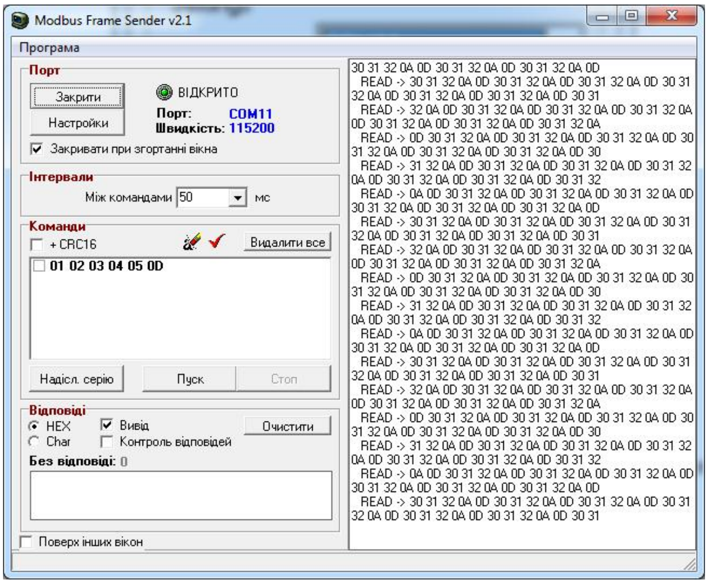


Рис. 5.35. Вікно виводу програми “Modbus FS”

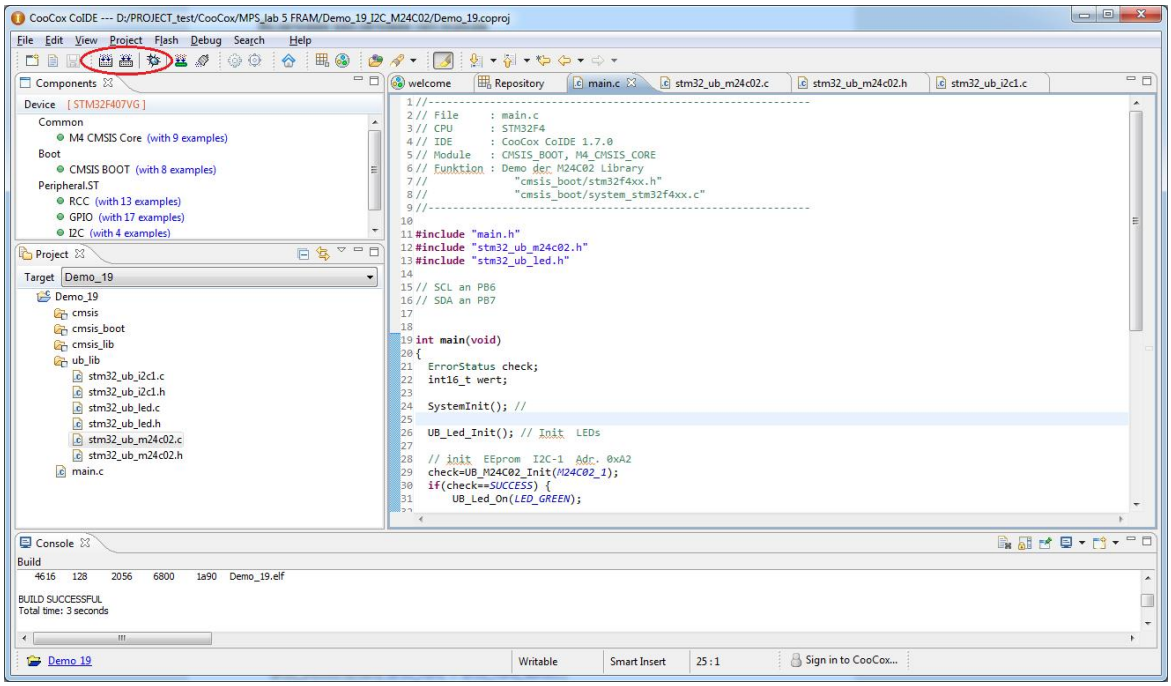


Рис. 5.36. Компіляція та запуск режиму відлагодження тестового проєкту CooCox звертання до FM24CL16

МЕТОДИЧНІ МАТЕРІАЛИ

Ядро Cortex-M4F розроблено для проєктування вбудованих мікрокомп'ютерних систем середньої продуктивності з низькою споживною потужністю. Архітектурно це 32-розрядне RISC-ядро типу ARM нового покоління з повним набором інструкцій ARM, реалізацією інструкцій DSP, вбудованим процесором плаваючої коми (FPU) та максимальною робочою частотою до 200 MHz. Ядро має вбудований модуль захисту пам'яті який підвищує безпеку роботи виконання програм. Для оптимального програмування FPU використовується спеціальна мета-мова. Все це дозволяє ефективно обробляти сигнали та реалізовувати складні алгоритми керування.

На базі ядра Cortex-M4F різні фірми освоїли випуск широкої номенклатури мікроконтролерів з різними об'ємами інтегрованої на кристалі пам'яті та різноманітною номенклатурою інтегрованих периферійних пристроїв. Одними з таких мікроконтролерів є мікроконтролери серії STM32F4xx фірми STMicroelectronics які стали досить популярними серед розробників.

Вказані мікроконтролери характеризуються наявністю інтегрованої швидкодіючої пам'яті Flash до 1 Mb та наявністю кеш-пам'яті що забезпечує звертання до пам'яті на максимальній частоті процесора без циклів очікування. Система обробки подій в реальному часі не вимагає втручання процесора що забезпечує обробку без затримок. Номенклатура периферійних пристроїв різних мікроконтролерів поряд з класичними інтерфейсними пристроями включає такі комунікаційні інтерфейси як Ethernet MAC з підтримкою стандарту IEEE1588, CAN, USB, інтерфейси для підключення відеокамери та графічних дисплеїв, 12 та 16-розрядні багатоканальні швидкодіючі аналого-цифрові перетворювачі (АЦП), спеціалізовані цифро-аналогові перетворювачі (ЦАП). Для забезпечення високої продуктивності при обміні даними на кристалах інтегровані канали

прямого доступу до пам’яті. Деякі мікроконтролери мають інтегровані на кристалі такі специфічні вузли як криптографічний процесор, модуль обчислення контрольних сум CRC, генератор випадкових чисел.

Важливою особливістю вказаних мікроконтролерів є можливість функціонування в режимі пониженого споживання при батарейному живленні. Також при пропаданні основного живлення забезпечується зберігання важливих даних в SRAM з батарейним живленням та функціонування таймера реального часу. Мікроконтролери випускаються в різних корпусах з різною кількістю виводів від 64 до 176, що дозволяє оптимізувати апаратні засоби при різних застосуваннях. Напруга живлення мікроконтролерів від 1.8V до 3.6V , температура в робочому режимі від – 40 до +105 градусів С.

ВНУТРІШНЯ СТРУКТУРА STM32407

Основні характеристики

• ядро 32 розряди;
• максимальна частота ядра 168 MHz;
• вбудований процесор плаваючої коми FPU;
• інструкції ARM та DSP;
• вбудована Flash пам’ять програм до 1Mb та SRAM даних до 192 Kb з можливістю зовнішнього розширення;
• SRAM 4Kb з зовн. резервним живленням;
• широка номенклатура інтегрованої периферії;
• VDD = 3.3 V (1.8 V ≤ VDD ≤ 3.6 V)

Вузли STM32F40x

• ядро ARM® Cortex™-M4F 168 MHz;
• система синхронізації PLL;
• інтегрований процесор FPU;
• інтегрована Flash пам’ять програм до 1Mb;
• інтегрована SRAM даних до 192 Kb;
• інтегрована резервна SRAM даних 4Kb (Vbat);
• адаптивний акселератор памя’ті (ART Accelerator);
• блок захисту пам’яті (MPU);
• модуль обчислення контрольних сумм (CRC unit);
• генератор випадкових чисел (RNG);
• мульти-AHB матриця шин 32 р;
• контролер прямого доступу до пам’яті (DMA);
• контролер статичної пам’яті (FSMC) з можливісю конфігурації керування PKI;
• контролер вкладених векторних переривань (NVIC);
• контролер зовнішніх переривань (EXTI);
• широка номенклатура інтерфейсних вузлів для паралельного та послідовного обміну (GPIO, (SDIO)-SD/SDIO/MMC, USART, SPI, I²C, CAN, USB, Ethernet, I2S);
• інтерфейс цифрової камери (DCMI);
• інтегровані 12р АЦП та ЦАП;
• універсальні таймери, RTC, WDT, SysTick ;
• вбудовані макро-регістри трасування (Embedded Trace Macrocell, ETM);
• JTAG та SW інтерфейси для трасування та програмування інтегрованої Flash;
• інтегрований стабілізатор напруги живлення;
• вузол керування живленням;

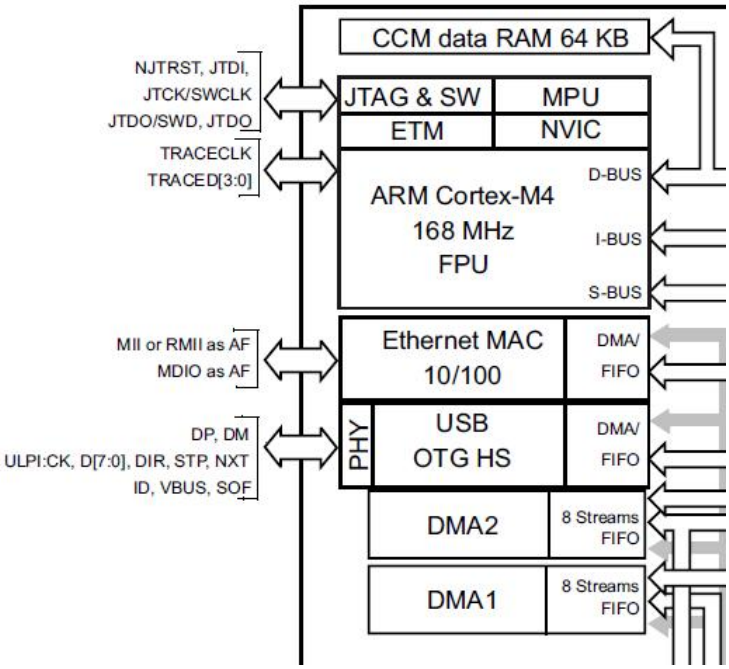


Рис. 5.37а. Внутрішня структура STM32F407

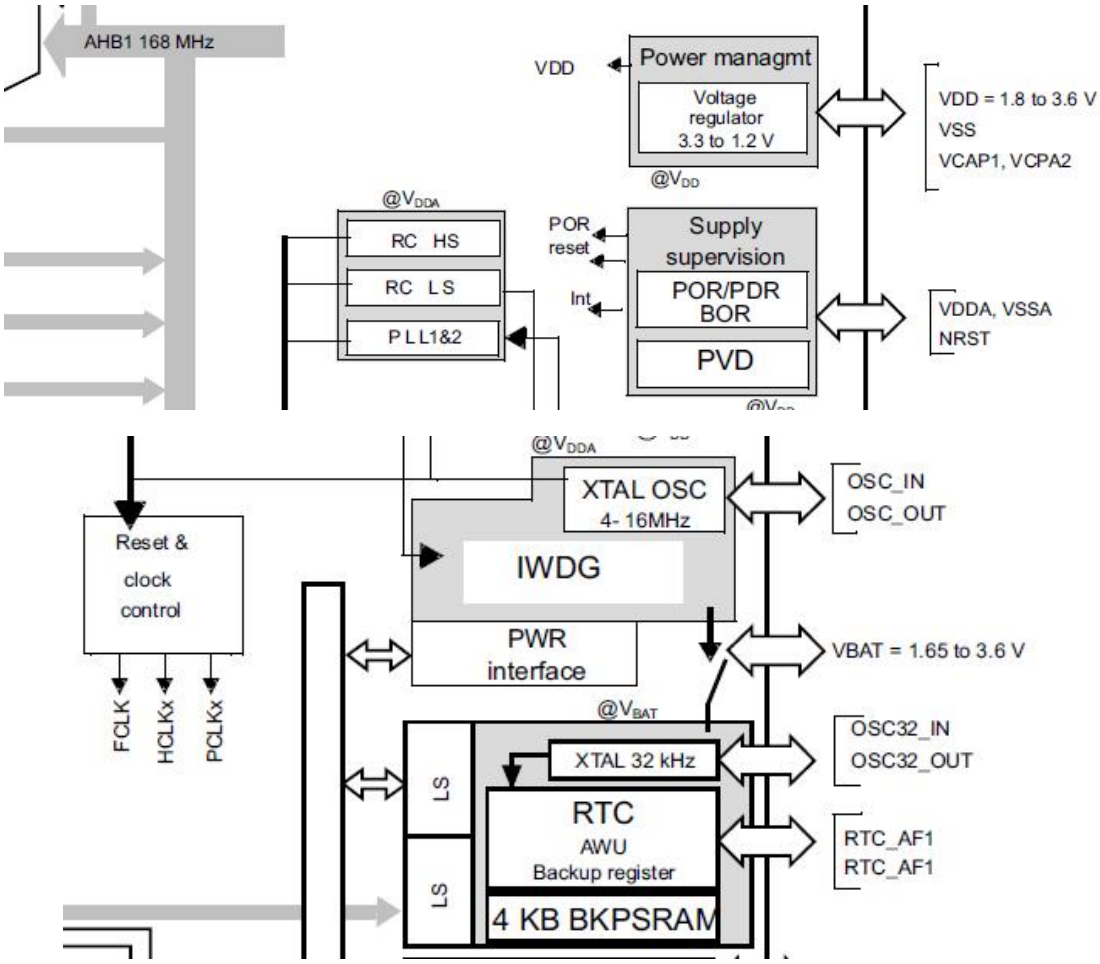


Рис. 5.37б. Внутрішня структура STM32F407

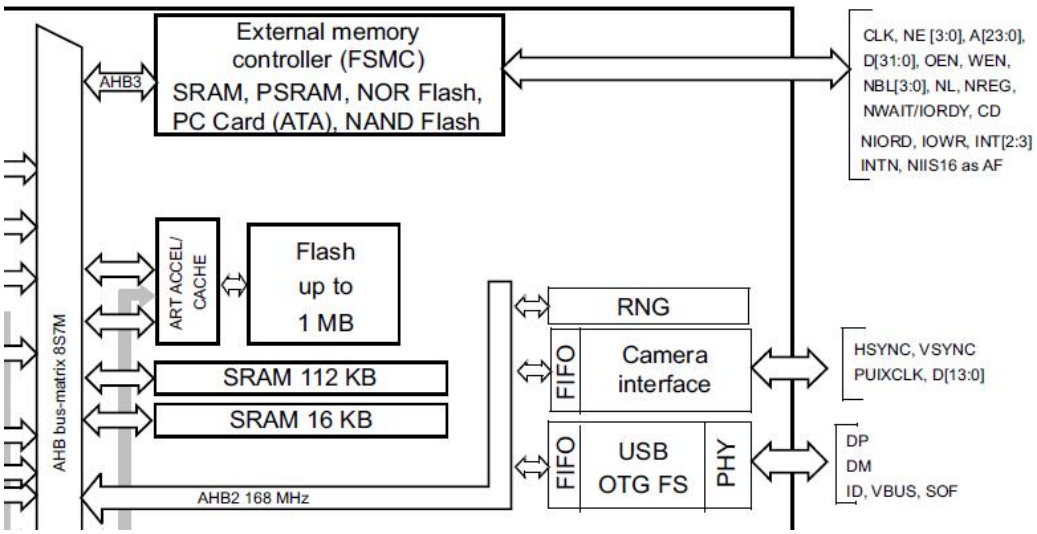


Рис. 5.37б (продовження). Внутрішня структура STM32F407

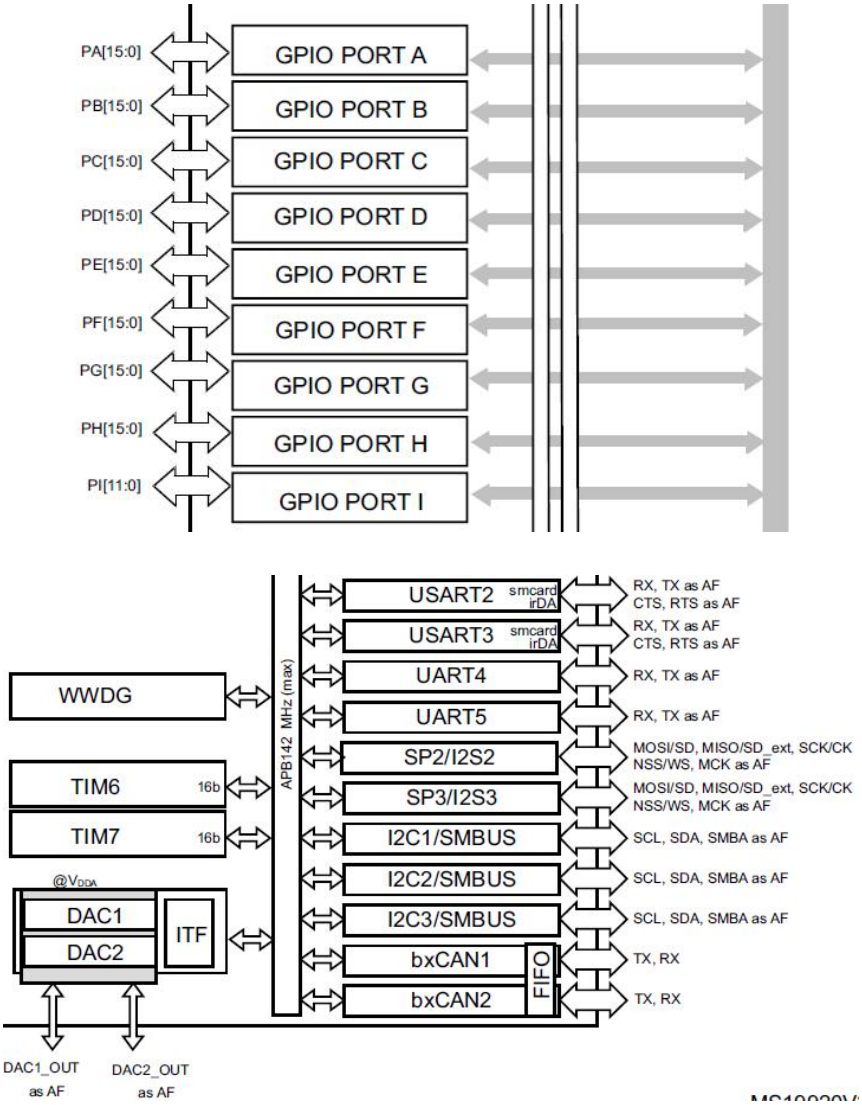


Рис. 5.37в. Внутрішня структура STM32F407

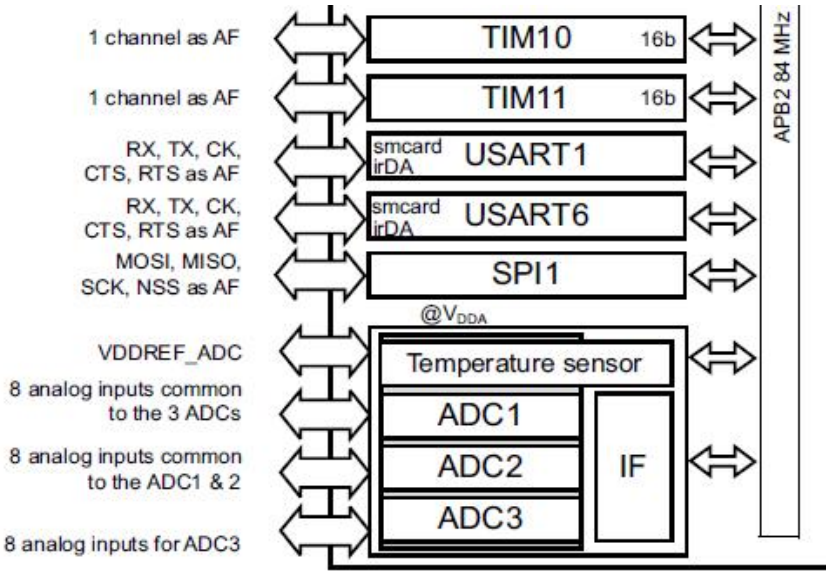


Рис. 5.37в (продовження). Внутрішня структура STM32F407

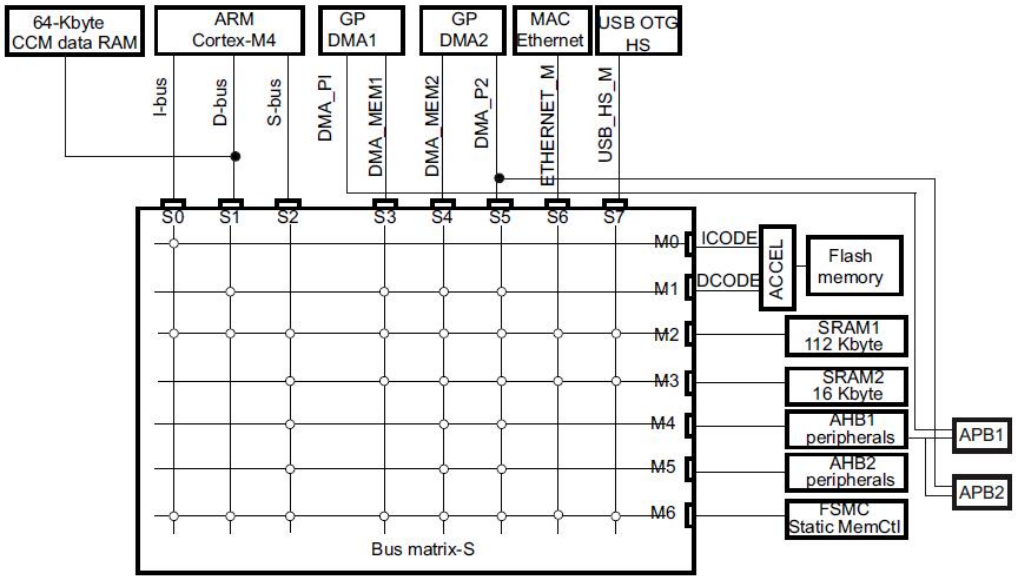


Рис. 5.38. Мульти-АНВ матриця шин 32 р

КПДП (DMA)
memory-to-memory, peripheral-to-memory and memory-to-peripheral

- SPI and I2S
- I2C
- USART
- General-purpose, basic and advanced-control timers TIMx
- DAC
- SDIO
- Camera interface (DCMI)
- ADC.

Живлення

• VDD = 1.8 to 3.6 V: external power supply for I/Os and the internal regulator (when enabled), provided externally through VDD pins.

• VSSA, VDDA = 1.8 to 3.6 V: external analog power supplies for ADC, DAC, Reset blocks, RCs and PLL. VDDA and VSSA must be connected to VDD and VSS, respectively.

• VBAT = 1.65 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when VDD is not present.

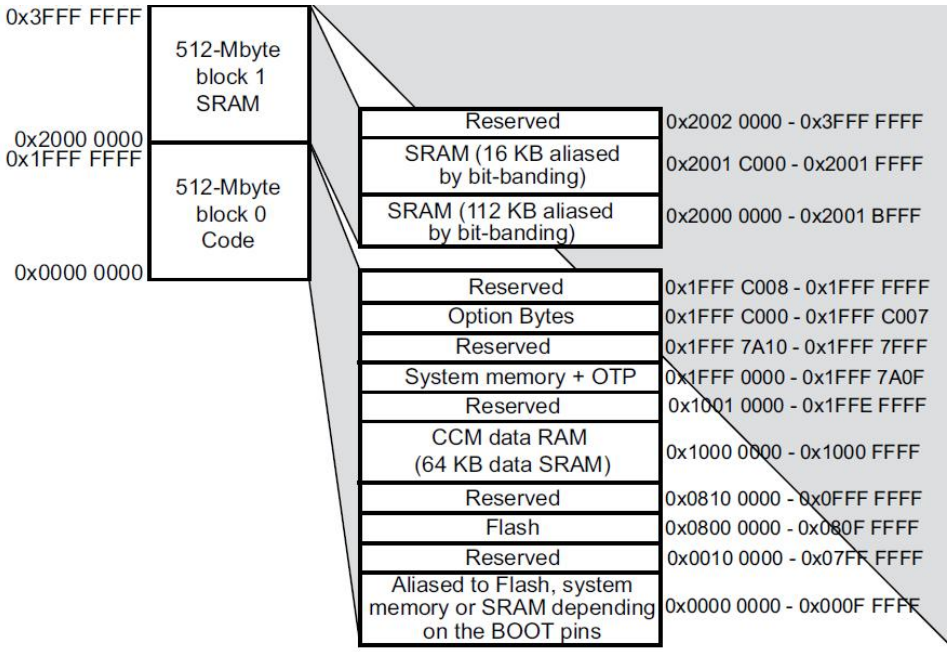


Рис. 5.39а. Структура пам'яті STM32F40x

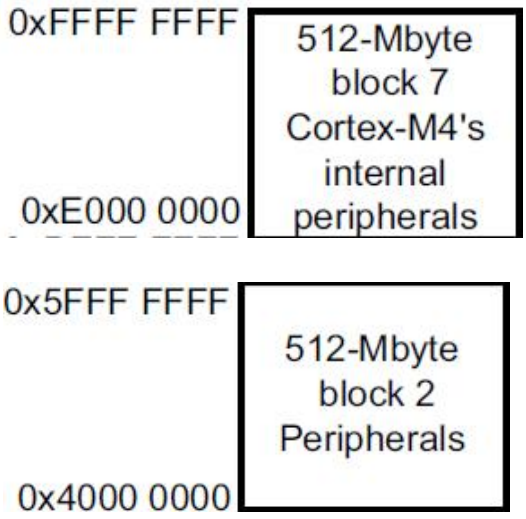


Рис. 5.39б. Структура пам'яті STM32F40x

Bus	Boundary address	Peripheral
	0xE00F FFFF - 0xFFFF FFFF	Reserved
Cortex-M4	0xE000 0000 - 0xE00F FFFF	Cortex-M4 internal peripherals
	0xA000 1000 - 0xDFFF FFFF	Reserved
AHB3	0xA000 0000 - 0xA000 0FFF	FSMC control register
	0x9000 0000 - 0x9FFF FFFF	FSMC bank 4
	0x8000 0000 - 0x8FFF FFFF	FSMC bank 3
	0x7000 0000 - 0x7FFF FFFF	FSMC bank 2
	0x6000 0000 - 0x6FFF FFFF	FSMC bank 1
	0x5006 0C00- 0x5FFF FFFF	Reserved
AHB2	0x5006 0800 - 0x5006 0BFF	RNG
	0x5005 0400 - 0x5006 07FF	Reserved
	0x5005 0000 - 0x5005 03FF	DCMI
	0x5004 0000- 0x5004 FFFF	Reserved
	0x5000 0000 - 0x5003 FFFF	USB OTG FS
	0x4008 0000- 0x4FFF FFFF	Reserved

Рис. 5.40а. Регістри STM32F40x

Bus	Boundary address	Peripheral
AHB1	0x4004 0000 - 0x4007 FFFF	USB OTG HS
	0x4002 9400 - 0x4003 FFFF	Reserved
	0x4002 9000 - 0x4002 93FF	ETHERNET MAC
	0x4002 8C00 - 0x4002 8FFF	
	0x4002 8800 - 0x4002 8BFF	
	0x4002 8400 - 0x4002 87FF	
	0x4002 8000 - 0x4002 83FF	
	0x4002 6800 - 0x4002 7FFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 5FFF	Reserved
	0x4002 4000 - 0x4002 4FFF	BKPSRAM
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2400 - 0x4002 2FFF	Reserved
	0x4002 2000 - 0x4002 23FF	GPIOI
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA
	0x4001 5800- 0x4001 FFFF	Reserved

Рис. 5.40б. Регістри STM32F40x

Регістри паралельних портів

GPIO Registers

Configuration Registers

STM32 містить чотири регістри конфігурації для кожного з портів:

- Port mode register – GPIOx_MODER
- Output type register – GPIOx_OTYPER
- Speed register – GPIOx_OSPEEDR
- Pull-up/Pull-down register – GPIOx_PUPDR

Кожен з цих регістрів має довжину 32 біти, хоча і не всі біти використовуються у всіх регістрах.

Port Mode Register – GPIOx_MODER

Цей 32-розрядний регістр має 2 бітні значення, які визначають режим роботи.
Можна встановити такі режими:

Bit Values	Description
00	Input
01	General purpose output
10	Alternate function
11	Analog

Port	Reset Value
A	0xA800 0000
B	0x0000 0280
All others	0x0000 0000

Output Type Register – GPIOx_OTYPER

Bit Value	Description
0	Push/pull output
1	Open drain output

Speed Register – GPIOx_OSPEEDR

Bit Values	Description
00	2 MHz Low speed
01	25 MHz Medium speed
10	50 MHz Fast speed
11	100 MHz High speed on 30pF 80 MHz High speed on 15 pF

Pull-up/Pull-down register – GPIOx_PUPDR

Bit Values	Description
00	No pull-up or pull-down resistor
01	Pull-up resistor
10	Pull-down resistor
11	Reserved

Input data register – GPIOx_IDR

Bit Value	Description
0	High logic value
1	Low logic value

Output data register – GPIOx_ODR

Bit Value	Description
0	High logic value
1	Low logic value

Вузол свідлодіодів модуля STM32F4DISCOVERY

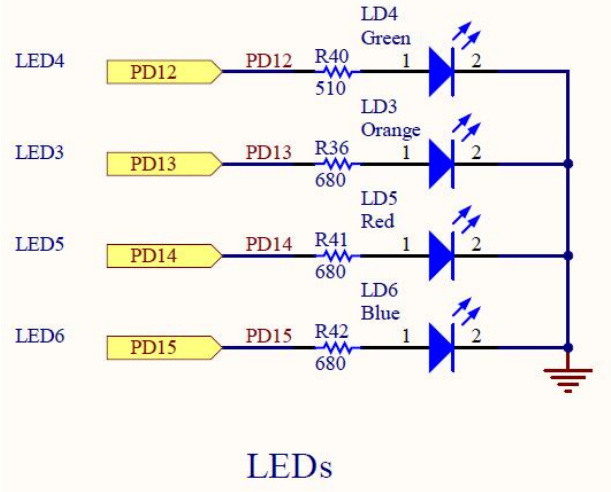


Рис. 5.41. Схема вузла світлодіодів модуля STM32F4DISCOVERY

Процедура ініціалізації порту PD до якого підключені світлодіоди

```
void UB_Led_Init (void)
{
  GPIO_InitTypeDef  GPIO_InitStructure;
  LED_NAME_t led_name;

    for(led_name=0;led_name<LED_ANZ;led_name++) {
    // Clock Enable
    RCC_AHB1PeriphClockCmd(LED[led_name].LED_CLK, ENABLE);

    // Config
    GPIO_InitStructure.GPIO_Pin = LED[led_name].LED_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
    GPIO_Init(LED[led_name].LED_PORT, &GPIO_InitStructure);

    // Default
    if(LED[led_name].LED_INIT==LED_OFF) {
          UB_Led_Off(led_name);
    }
    else {
         UB_Led_On(led_name);
    }
  }
}
```

Включення-виключення світлодіодів

```
void UB_Led_Off(LED_NAME_t led_name)
{
  LED[led_name].LED_PORT->BSRRH = LED[led_name].LED_PIN;
}

void UB_Led_On(LED_NAME_t led_name)
{
  LED[led_name].LED_PORT->BSRRL = LED[led_name].LED_PIN;
}
```

Вузол реле лабораторного стенда

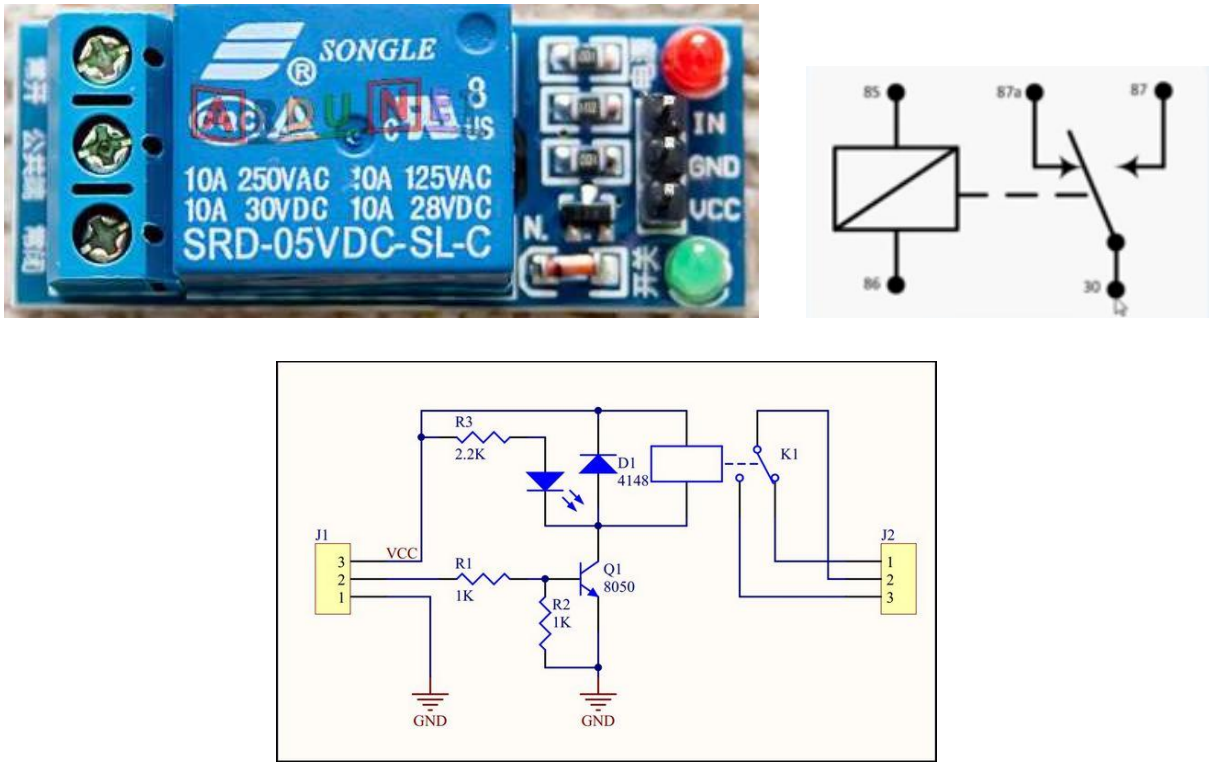


Рис. 5.42. Вузол реле лабораторного стенду

Лінія керування реле J1/2 підключається до лінії 2 порту E

Ініціалізація лінії порту E.2

```
/* Configure PE2 in output pushpull mode */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
```

Керування реле

```
  GPIO_WriteBit(GPIOE,GPIO_Pin_2,Bit_RESET); //RL On
  GPIO_WriteBit(GPIOE,GPIO_Pin_2,Bit_SET);   //RL Off
```

Вузол RS-485

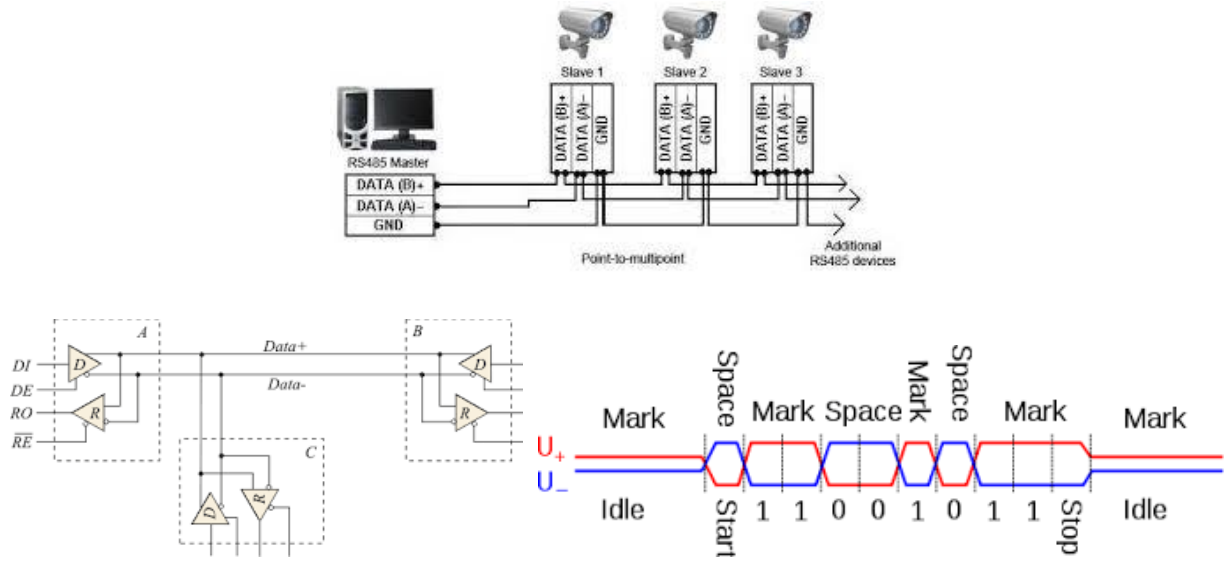


Рис. 5.43. Інтерфейс RS-485

Обмін інформацією між модулем STM32F4DISCOVERY та ПК здійснюється через порт UART2 мікроконтролера STM32F407 модуля STM32F4DISCOVERY та адаптер USB-RS485 підключений до ПК.

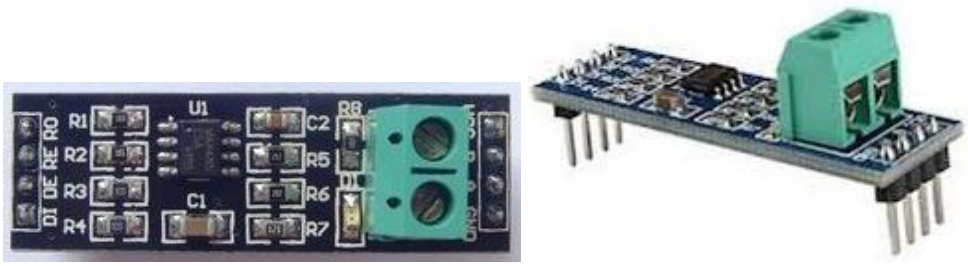


Рис. 5.44. Зовнішній вигляд вузла RS-485 лабораторного стенду

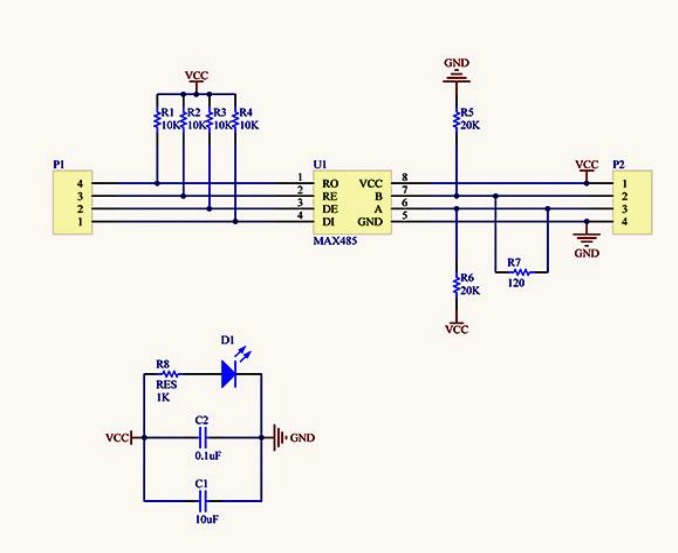


Рис. 5.45. Схема вузла RS-485 лабораторного стенду

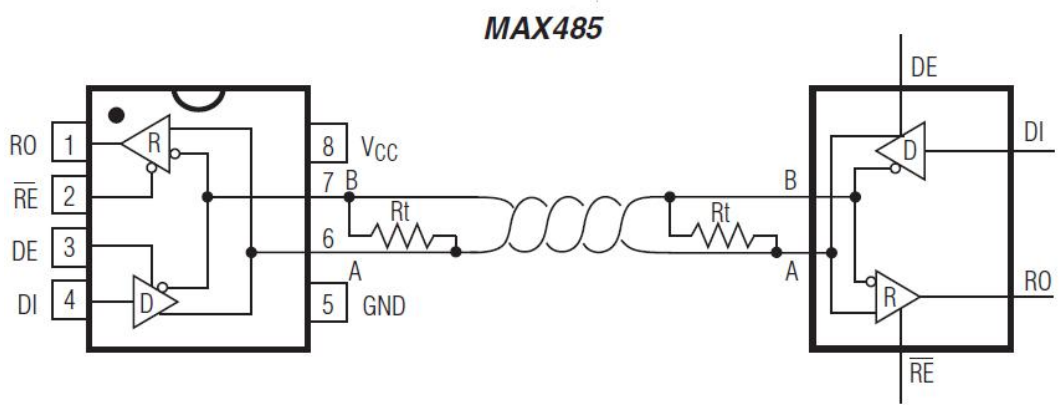


Рис. 5.46. Мікросхема MAX485 перетворювача UART-RS485

Рис. 5.47. Зовнішній вигляд адаптера USB-RS485

Входи RE/DE м/сх MAX485 керуються лінією 1 порту A.

Ініціалізація лінії керування:

```
/* Configure PA1 in output pushpull mode */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
```

```
  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Включення режиму “передача” модуль→ПК:

```
 GPIO_WriteBit(GPIOA,GPIO_Pin_1,Bit_SET); //RS-485
```

Включення режиму “читання” модуль←ПК:

```
 GPIO_WriteBit(GPIOA,GPIO_Pin_1,Bit_SET); //RS-485
```

Ініціалізація порту UART2:

```
void UB_Uart_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;
  UART_NAME_t nr;

  for(nr=0;nr<UART_ANZ;nr++) {

    // Clock enable TX  RX Pins
    RCC_AHB1PeriphClockCmd(UART[nr].TX.CLK, ENABLE);
    RCC_AHB1PeriphClockCmd(UART[nr].RX.CLK, ENABLE);

    // Clock enable UART
    if((UART[nr].UART==USART1) || (UART[nr].UART==USART6)) {
      RCC_APB2PeriphClockCmd(UART[nr].CLK, ENABLE);
    }
    else {
      RCC_APB1PeriphClockCmd(UART[nr].CLK, ENABLE);
    }

    // UART Alternative-Funktions  IO-Pins
    GPIO_PinAFConfig(UART[nr].TX.PORT,UART[nr].TX.SOURCE,UART[nr].AF);
    GPIO_PinAFConfig(UART[nr].RX.PORT,UART[nr].RX.SOURCE,UART[nr].AF);

    // UART Alternative-Funktion  PushPull
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    // TX-Pin
    GPIO_InitStructure.GPIO_Pin = UART[nr].TX.PIN;
```

```
    GPIO_Init(UART[nr].TX.PORT, &GPIO_InitStructure);
    // RX-Pin
    GPIO_InitStructure.GPIO_Pin =  UART[nr].RX.PIN;
    GPIO_Init(UART[nr].RX.PORT, &GPIO_InitStructure);


    USART_OverSampling8Cmd(UART[nr].UART, ENABLE);

    // init Baudrate, 8Databits, 1Stopbit, No Paritaet, No RTS+CTS
    USART_InitStructure.USART_BaudRate = UART[nr].BAUD;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(UART[nr].UART, &USART_InitStructure);

    // UART enable
    USART_Cmd(UART[nr].UART, ENABLE);

    // RX-Interrupt enable
    USART_ITConfig(UART[nr].UART, USART_IT_RXNE, ENABLE);

    // enable UART Interrupt-Vector
    NVIC_InitStructure.NVIC_IRQChannel = UART[nr].INT;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // RX-Puffer
    UART_RX[nr].rx_buffer[0]=RX_END_CHR;
    UART_RX[nr].wr_ptr=0;
    UART_RX[nr].rd_ptr=0;
    UART_RX[nr].status=RX_EMPTY;
  }
}
```

Передача інформації з буфера “buf” через UART в ПК:

```
UB_Uart_SendString(COM2,buf,LFCR);
```

FM24CL16

16K bit Ferroelectric Nonvolatile RAM
Organized as 2,048 x 8 bits
Unlimited Read/Write Cycles
10 year Data Retention
NoDelay™ Writes
Advanced High-Reliability Ferroelectric Process

Fast Two-wire Serial Interface
Up to 1MHz maximum bus frequency
Direct hardware replacement for EEPROM

Low Power Operation
New 2.7 – 3.6V operation
5 mkA Active Current (100 kHz) @ 3V
1 mkA Standby Current

Industry Standard Configuration
Industrial Temperature -40 град.C to +85град.C
8-pin SOIC

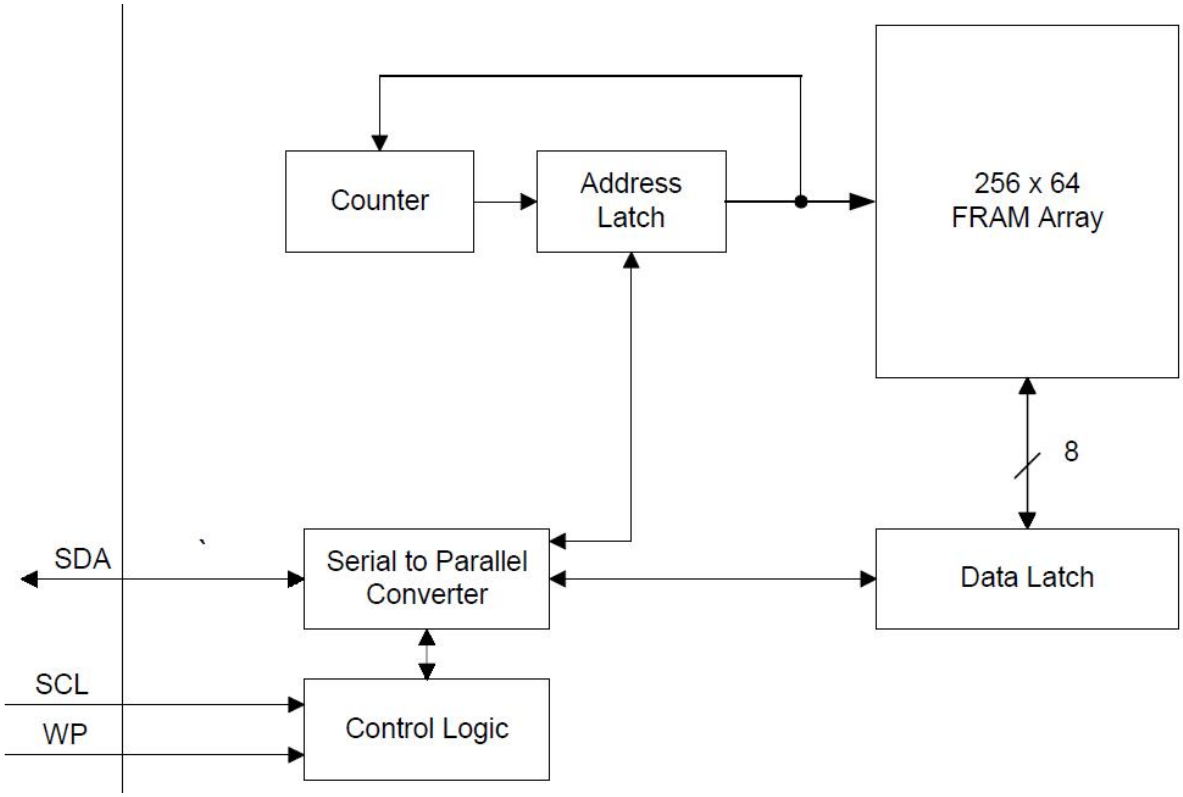
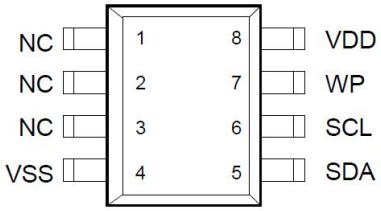


Рис. 5.48. Внутрішня структура FM24CL16



Pin Names	Function
SDA	Serial Data/Address
SCL	Serial Clock
WP	Write Protect
VDD	Supply Voltage
VSS	Ground

Таблиця 5.1. Сигнали інтерфейсу I2C

Pin Name	Type	Pin Description
SDA	I/O	Serial Data Address: This is a bi-directional data pin for the two-wire interface. It employs an open-drain output and is intended to be wire-OR'd with other devices on the two-wire bus. The input buffer incorporates a Schmitt trigger for noise immunity and the output driver includes slope control for falling edges. A pull-up resistor is required.
SCL	Input	Serial Clock: The serial clock input for the two-wire interface. Data is clocked-out on the falling edge and clocked-in on the rising edge.
WP	Input	Write Protect: When WP is high, the entire array is write-protected. When WP is low, all addresses may be written. This pin is internally pulled down.
VDD	Supply	Supply Voltage (3V)
VSS	Supply	Ground
NC	-	No connect

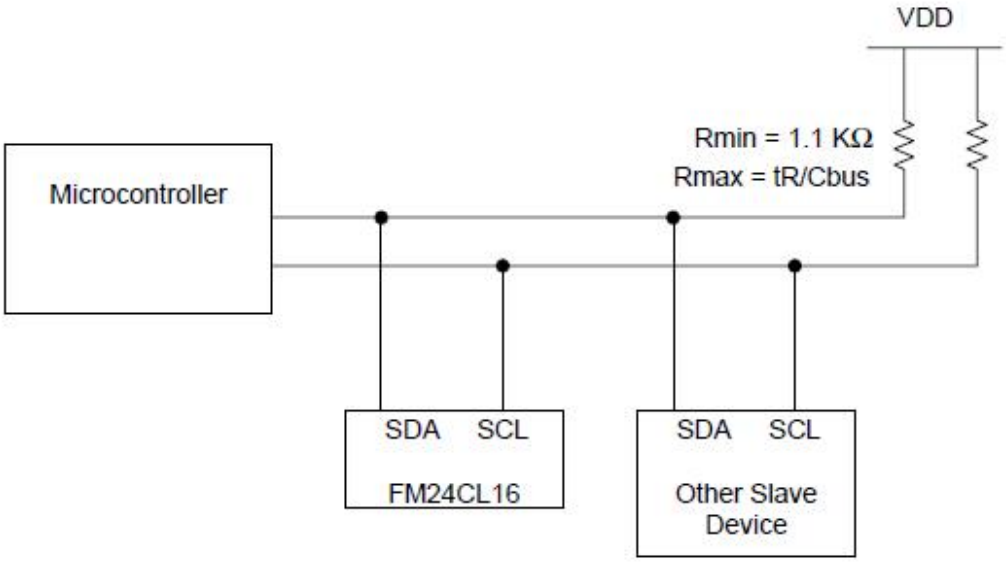


Рис. 5.49. Підключення FM24CL16 до мікроконтролера

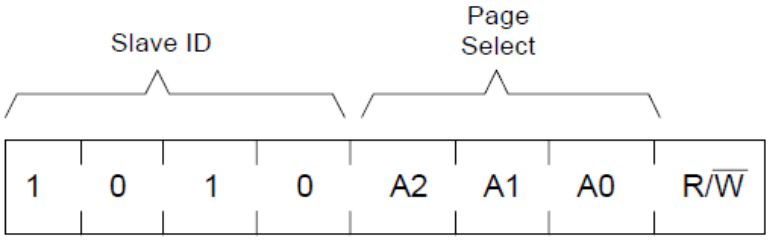


Рис. 5.50. Адресація FM24CL16

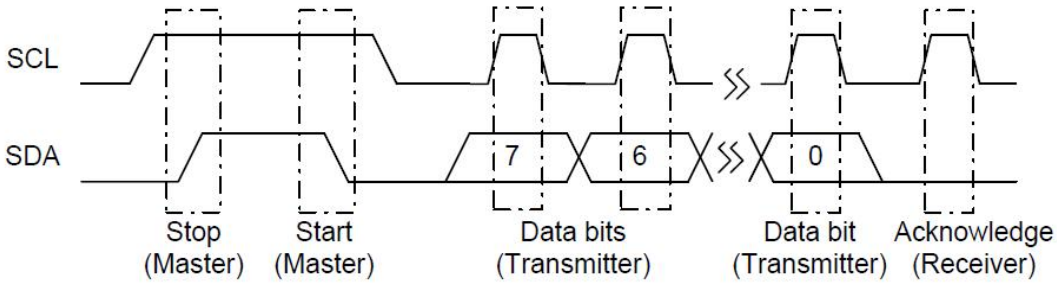


Рис. 5.51. Діаграма запису даних в FM24CL16

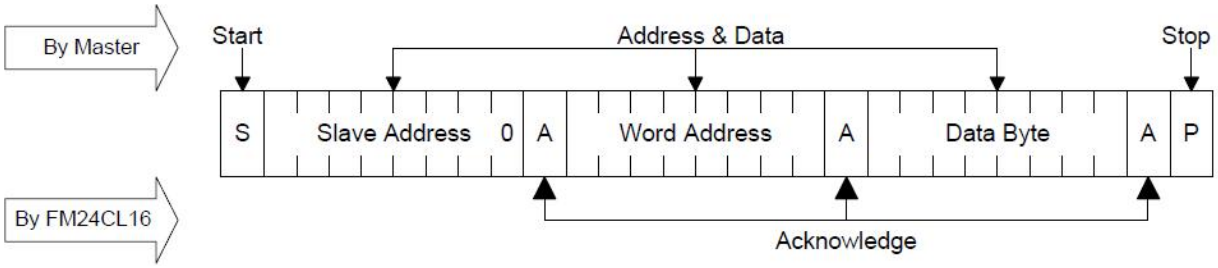


Рис. 5.52. Запис одиночного байта в FM24CL16

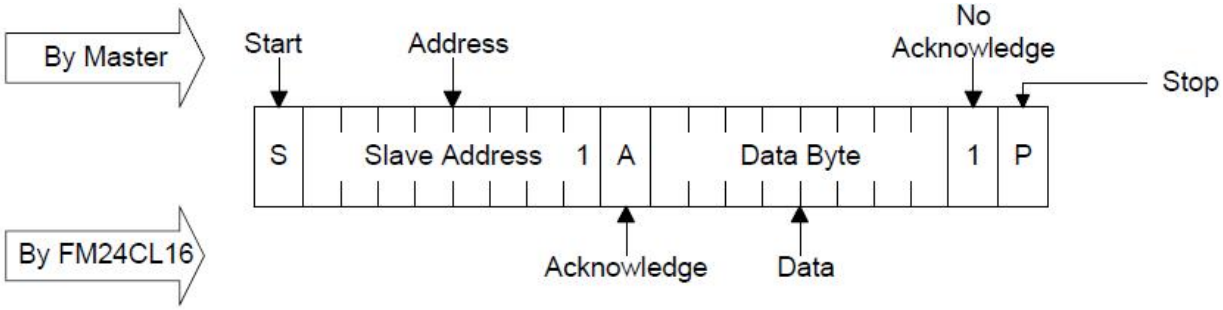


Рис. 5.53. Читання одиночного байта з FM24CL16

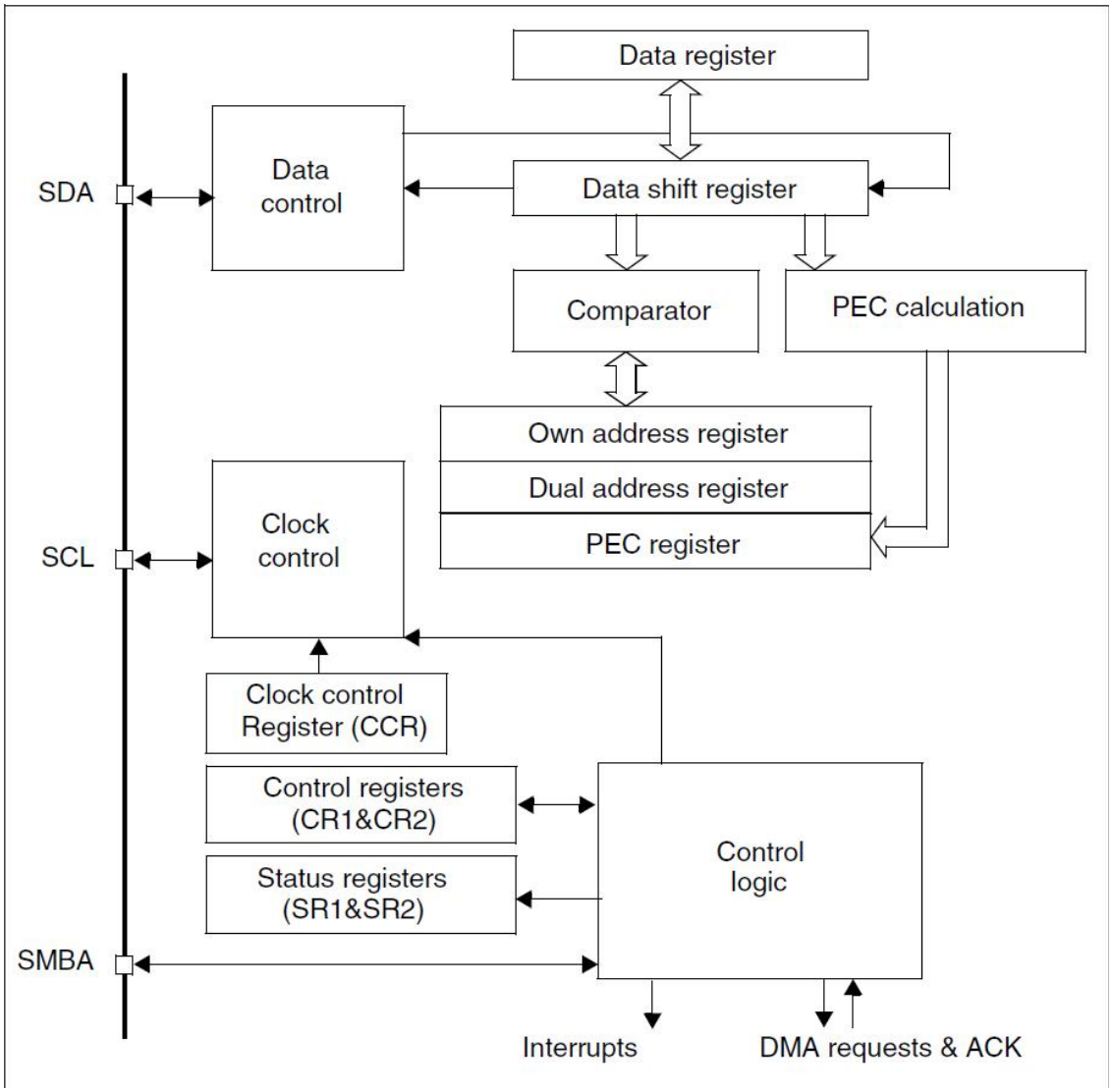


Рис. 5.54. Внутрішня структура вузла I2C STM32F407

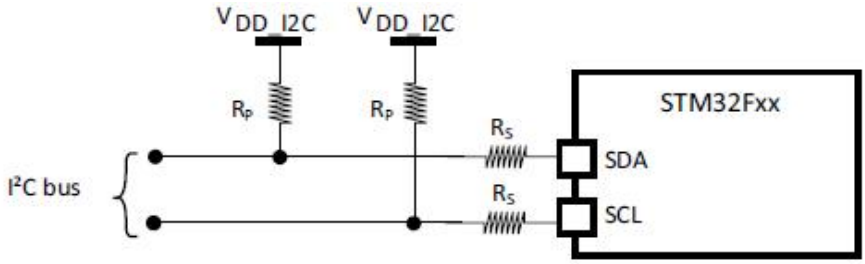
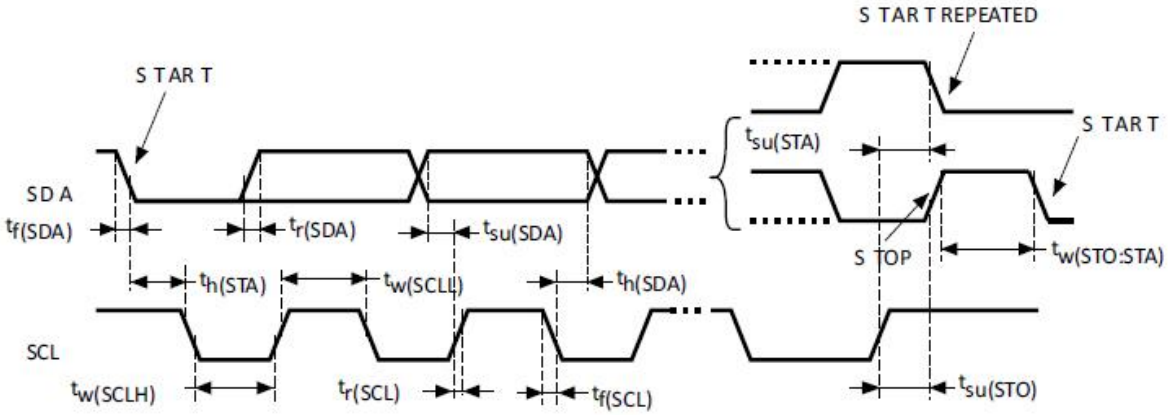


Рис. 5.55а. Лінії I2C1 STM32F407

PB6	I/O	FT		I2C1_SCL/ TIM4_CH1 / CAN2_TX / DCMI_D5/USART1_TX/ EVENTOUT
PB7	I/O	FT		I2C1_SDA / FSMC_NL / DCMI_VSYNC / USART1_RX/ TIM4_CH2/ EVENTOUT

Рис. 5.55б. Лінії I2C1 STM32F407



Symbol	Parameter	Standard mode I^2C(1)		Fast mode I^2C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{h(STA)}$	Start condition hold time	4.0	-	0.6	-	µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	0.6	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	0.6	-	µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7	-	1.3	-	µs
C_b	Capacitive load for each bus line	-	400	-	400	pF

Рис. 5.56. Часова діаграма вузла I2C STM32F407

Таблиця 5.2. Регістри I2C STM32F407

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	I2C_CR1	Reserved																SWRST	Reserved	ALERT	PEC	POS	ACK	STOP	START	NOSTRETCH	ENGC	ENPEC	ENARP	SMBTYPE	Reserved	SMBUS	PE
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0		0	0
0x04	I2C_CR2	Reserved																			LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved	FREQ[5:0]						
	Reset value																				0	0	0	0	0		0	0	0	0	0	0	0
0x08	I2C_OAR1	Reserved																ADDMODE	Reserved					ADD[9:8]		ADD[7:1]							ADD0
	Reset value																	0						0	0	0	0	0	0	0	0	0	0
0x0C	I2C_OAR2	Reserved																								ADD2[7:1]							ENDUAL
	Reset value																									0	0	0	0	0	0	0	0
0x10	I2C_DR	Reserved																								DR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	I2C_SR1	Reserved																SMBALERT	TIMEOUT	Reserved	PECERR	OVR	AF	ARLO	BERR	TxE	RxNE	Reserved	STOPF	ADD10	BTF	ADDR	SB
	Reset value																	0	0		0	0	0	0	0	0	0		0	0	0	0	0
0x18	I2C_SR2	Reserved																PEC[7:0]								DUALF	SMBHOST	SMBDEFAULT	GENCALL	Reserved	TRA	BUSY	MSL
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x1C	I2C_CCR	Reserved																F/S	DUTY	Reserved	CCR[11:0]												
	Reset value																	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	I2C_TRISE	Reserved																										TRISE[5:0]					
	Reset value																											0	0	0	0	1	0
0x24	I2C_FLTR	Reserved																											ANOFF	DNF[3:0]			
	Reset value																												0	0	0	0	0

Література

5.1. STM32F405_07.pdf. STMicroelectronics.

5.2. Reference manual_en.DM00031020.pdf. STMicroelectronics.

5.3. STM32F4DISCOVERY.pdf. STMicroelectronics.

5.4. FM24CL16-G-ETC.pdf

5.5. MAX485.pdf

5.6. Конспект лекцій з дисципліни “Мікропроцесорні системи”

Процедура ініціалізації I2C1

```
// SCL PB6
// SDA PB7

Void UB_I2C1_Init(void)
{
 static uint8_t init_ok=0;
 GPIO_InitTypeDef  GPIO_InitStructure;

 // I2C-Clock enable
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

 // Clock Enable Pins
 RCC_AHB1PeriphClockCmd(I2C1DEV.SCL.CLK, ENABLE);
 RCC_AHB1PeriphClockCmd(I2C1DEV.SDA.CLK, ENABLE);

 // I2C reset
 RCC_APB1PeriphResetCmd(RCC_APB1Periph_I2C1, ENABLE);
 RCC_APB1PeriphResetCmd(RCC_APB1Periph_I2C1, DISABLE);

 // I2C Alternative-Funktions IO-Pins
 GPIO_PinAFConfig(I2C1DEV.SCL.PORT, I2C1DEV.SCL.SOURCE, GPIO_AF_I2C1);
 GPIO_PinAFConfig(I2C1DEV.SDA.PORT, I2C1DEV.SDA.SOURCE, GPIO_AF_I2C1);

 // I2C Alternative-Funktion
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
 GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;

 // SCL-Pin
 GPIO_InitStructure.GPIO_Pin = I2C1DEV.SCL.PIN;
 GPIO_Init(I2C1DEV.SCL.PORT, &GPIO_InitStructure);
 // SDA-Pin
 GPIO_InitStructure.GPIO_Pin = I2C1DEV.SDA.PIN;
 GPIO_Init(I2C1DEV.SDA.PORT, &GPIO_InitStructure);

 // I2C init
 P_I2C1_InitI2C();
}
```

Процедура запису байта

```
int16_t UB_I2C1_WriteByte(uint8_t slave_adr, uint8_t adr, uint8_t wert)
{
 int16_t ret_wert=0;
 uint32_t timeout=I2C1_TIMEOUT;
```

```
  // Start
  I2C_GenerateSTART(I2C1, ENABLE);

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-1));
  }

  // Slave-Adresse (write)
  I2C_Send7bitAddress(I2C1, slave_adr, I2C_Direction_Transmitter);

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_ADDR)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-2));
  }

  // ADDR-Flag
  I2C1->SR2;

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_TXE)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-3));
  }

  // Adresse write
  I2C_SendData(I2C1, adr);

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_TXE)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-4));
  }

  // Daten write
  I2C_SendData(I2C1, wert);

  timeout=I2C1_TIMEOUT;
  while ((!I2C_GetFlagStatus(I2C1, I2C_FLAG_TXE)) || (!I2C_GetFlagStatus(I2C1, I2C_FLAG_BTF))) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-5));
  }

  // Stop-
  I2C_GenerateSTOP(I2C1, ENABLE);

  ret_wert=0; // ok

  return(ret_wert);
}
```

Процедура читання байта

```
int16_t UB_I2C1_ReadByte(uint8_t slave_adr, uint8_t adr)
{
 int16_t ret_wert=0;
 uint32_t timeout=I2C1_TIMEOUT;

 // Start-
 I2C_GenerateSTART(I2C1, ENABLE);

 timeout=I2C1_TIMEOUT;
 while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB)) {
  if(timeout!=0) timeout--; else return(P_I2C1_timeout(-1));
 }

 // ACK disable
 I2C_AcknowledgeConfig(I2C1, DISABLE);

 // Slave-Adresse (write)
 I2C_Send7bitAddress(I2C1, slave_adr, I2C_Direction_Transmitter);

 timeout=I2C1_TIMEOUT;
 while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_ADDR)) {
  if(timeout!=0) timeout--; else return(P_I2C1_timeout(-2));
 }

 // ADDR-Flag
 I2C1->SR2;

 timeout=I2C1_TIMEOUT;
 while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_TXE)) {
  if(timeout!=0) timeout--; else return(P_I2C1_timeout(-3));
 }

 // Adresse write
 I2C_SendData(I2C1, adr);

 timeout=I2C1_TIMEOUT;
 while ((!I2C_GetFlagStatus(I2C1, I2C_FLAG_TXE)) || (!I2C_GetFlagStatus(I2C1, I2C_FLAG_BTF))) {
  if(timeout!=0) timeout--; else return(P_I2C1_timeout(-4));
 }

 // Start- write
 I2C_GenerateSTART(I2C1, ENABLE);

 timeout=I2C1_TIMEOUT;
 while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB)) {
  if(timeout!=0) timeout--; else return(P_I2C1_timeout(-5));
 }
```

```
  // Slave-Adresse write (read)
  I2C_Send7bitAddress(I2C1, slave_adr, I2C_Direction_Receiver);

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_ADDR)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-6));
  }

  // ADDR-Flag
  I2C1->SR2;

  timeout=I2C1_TIMEOUT;
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_RXNE)) {
    if(timeout!=0) timeout--; else return(P_I2C1_timeout(-7));
  }

  // Stop-
  I2C_GenerateSTOP(I2C1, ENABLE);

  // Daten
  ret_wert=(int16_t)(I2C_ReceiveData(I2C1));

  // ACK enable
  I2C_AcknowledgeConfig(I2C1, ENABLE);

  return(ret_wert);
}
```

Лабораторна робота № 6

ДОСЛІДЖЕННЯ ЦИФРОВОГО СИНТЕЗАТОРА ЧАСТОТИ В МПС

МЕТА РОБОТИ: ознайомлення з технічними характеристиками, архітектурою, системою команд мікроконтролера ADuC7128/7129, з основними режимами формування сигналу цифро-аналоговим перетворювачем ADuC7128/7129 та цифровим синтезатором частоти(DDS), режимами роботи інтерфейсу SPI.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Перевірити свою теоретичну підготовку по контрольних питаннях для самоперевірки; при необхідності скористатися методичними матеріалами.
2. Запустити систему mVision (**UVx.EXE**).
3. Ознайомитись з порядком створення проєкту, елементами екрану відлагоджувача, використанням команд головного меню та підменю, режимами відображення завантаженої програми, операціями модифікації вмісту комірок пам'яті та регістрів.
4. Ознайомитися з структурою ADuC7128/29, особливостями ядра ARM7, організацією пам'яті і системою команд.
5. Завантажити тестову програму, запустити покрокове виконання та проконтролювати процес формування сигналу цифро-аналоговим перетворювачем ADuC7128/29.
6. Виконати індивідуальне завдання визначене керівником лабораторної роботи.
7. Захистити лабораторну роботу.

ЗАВДАННЯ

1. В покроковому режимі продемонструвати ініціалізацію DAC та цифрового синтезатора частоти(DDS).
2. В покроковому режимі продемонструвати ініціалізацію інтерфейсу SPI
3. В покроковому режимі продемонструвати режими роботи інтерфейсу SPI
4. Продемонструвати формування сигналу цифро-аналоговим перетворювачем DAC.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1.Внутрішня структура мікроконтролера ADuC7128/29.
2.Структура процесорного ядра ARM7TDMI.
3.Особливості інструкцій.
4.Організація пам'яті мікроконтролера ADuC7128/29.
5.Структура DAC ADuC7128/29.
6.Порт SPI ADuC7128/29.
7.Основні етапи створення проєкту та опції меню відлагоджувача.

ОСНОВНІ ЕТАПИ ВИКОНАННЯ РОБОТИ

Завантажити відлагоджувальне середовище Keil mVision
Створення нового проєкту:

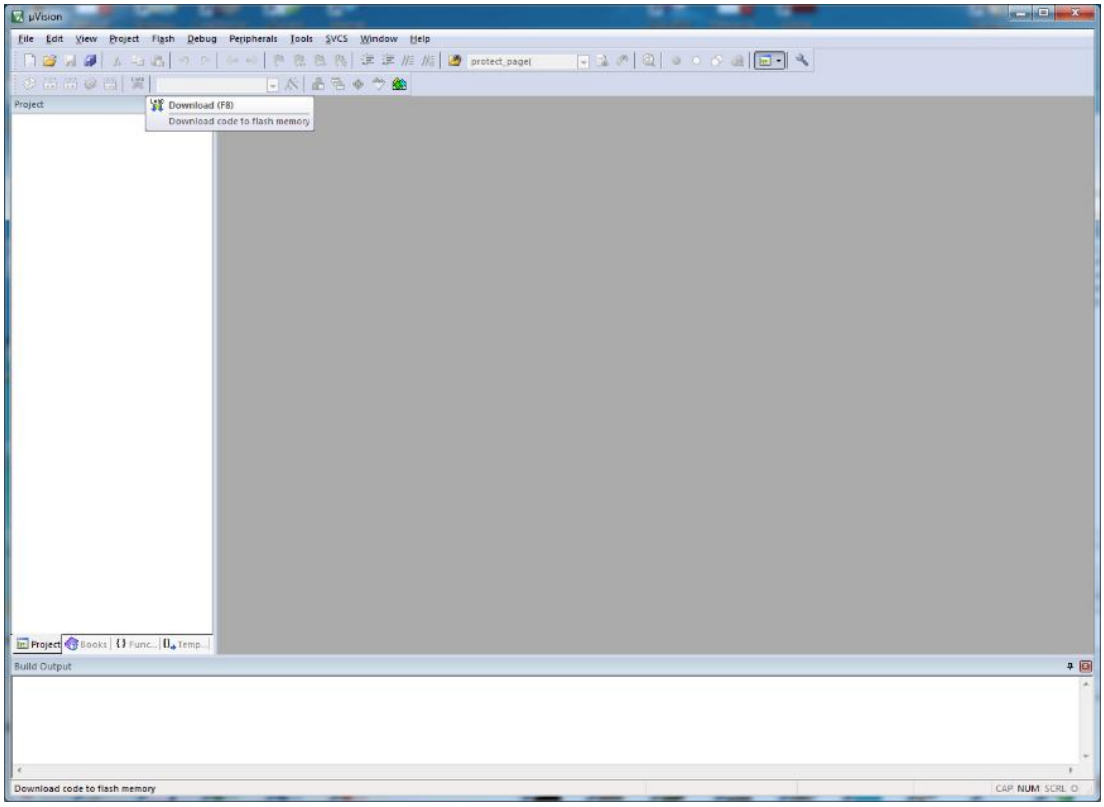


Рис. 6.1. Стартове вікно

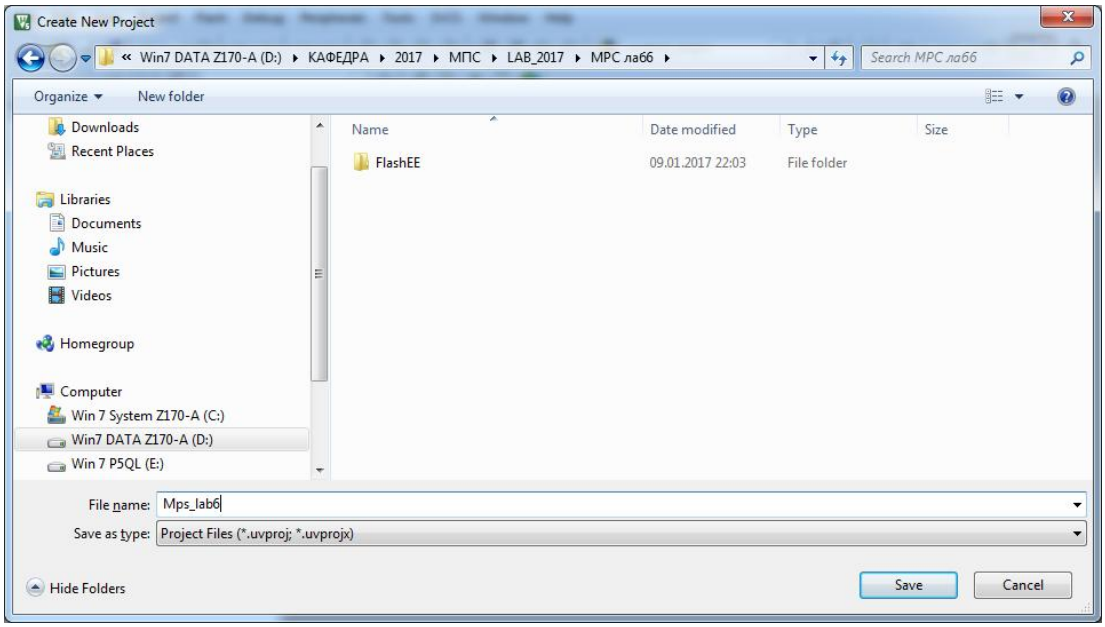


Рис. 6.2. Створення нового проєкту

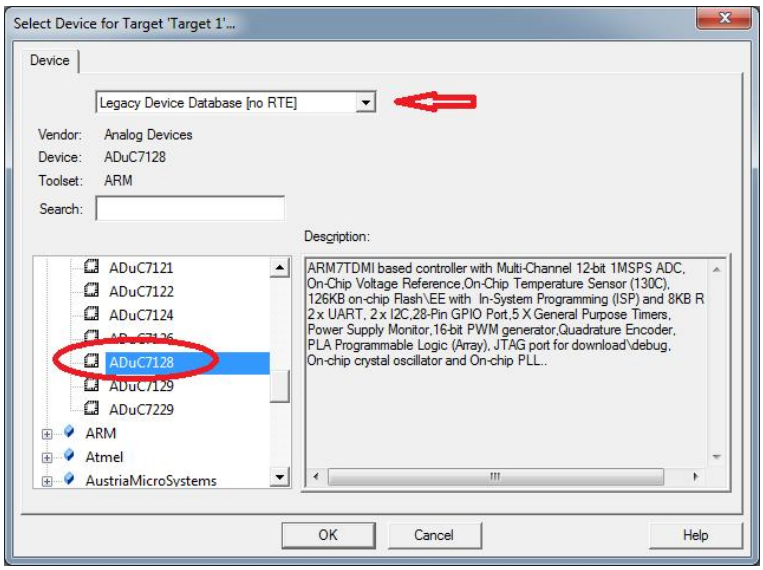


Рис. 6.3. Вибір мікроконтролера

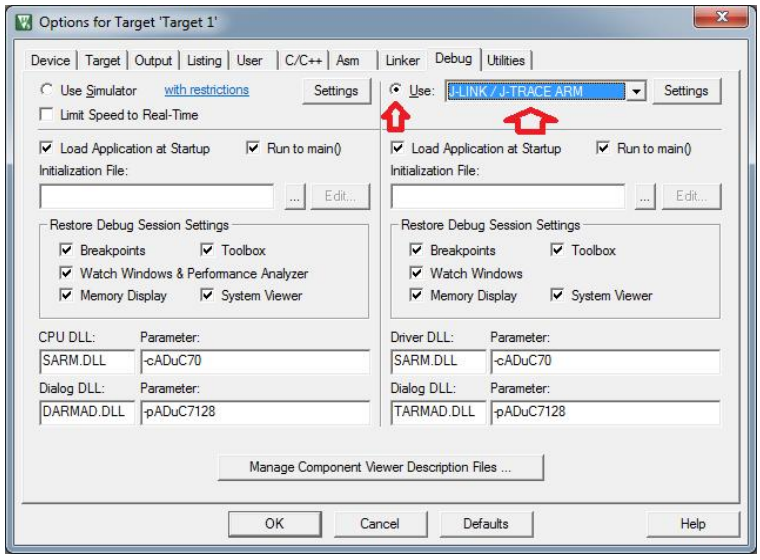


Рис. 6.4. Вибір адаптера відлагоджувача

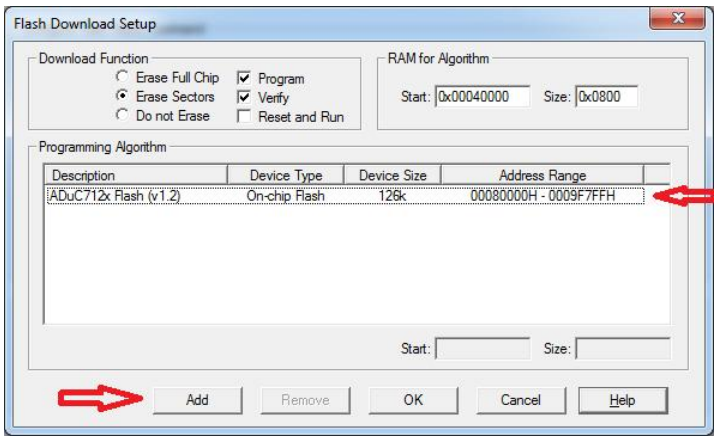


Рис. 6.5. Встановлення конфігурації програматора Flash

Вибравши піктограму Create new file в директорії проєкту створюємо файл main.c
За допомогою команди Add Existing Files to Group додаємо створений файл до проєкту

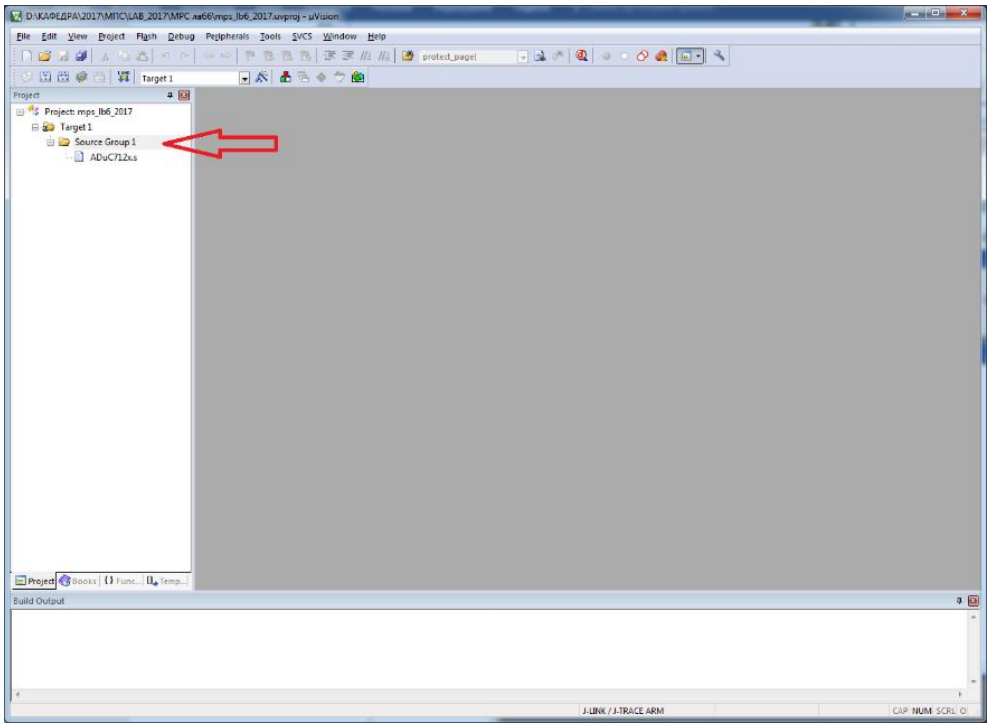


Рис. 6.6. Add Existing Files to Group

Відкрити (Project/Open Project) тестовий проєкт DAC, скомпілювати проєкт та запустити відлагоджувач. “d”:

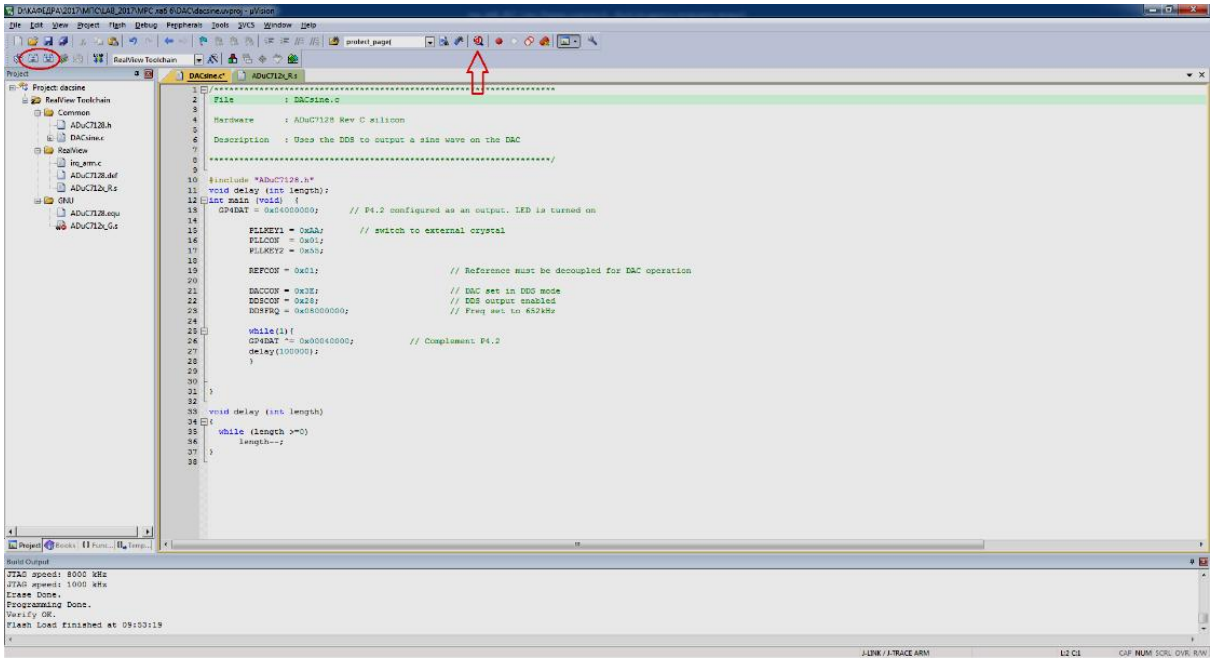


Рис. 6.7. Компіляція проєкту та запуск відлагоджувача

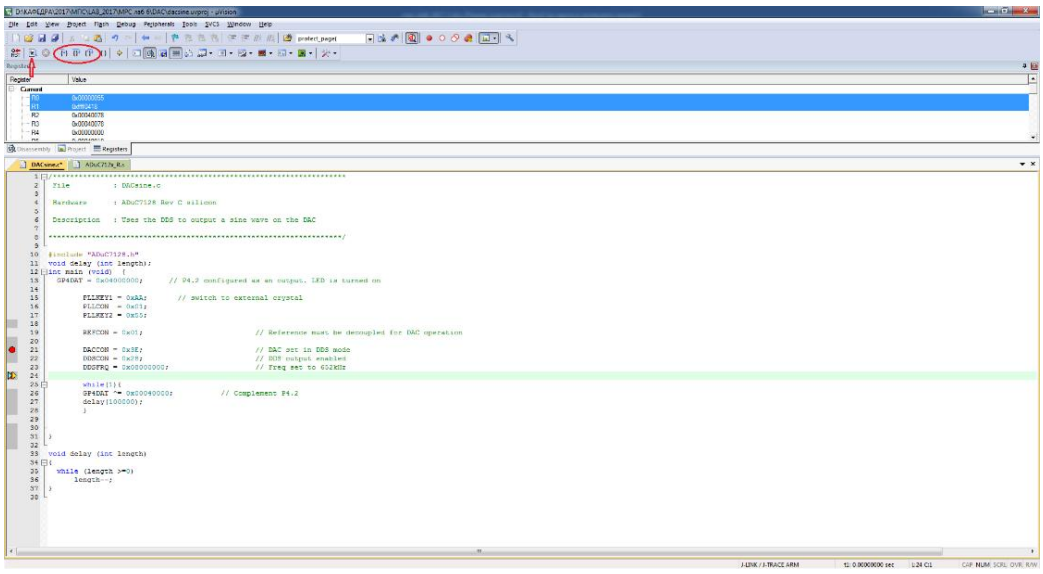
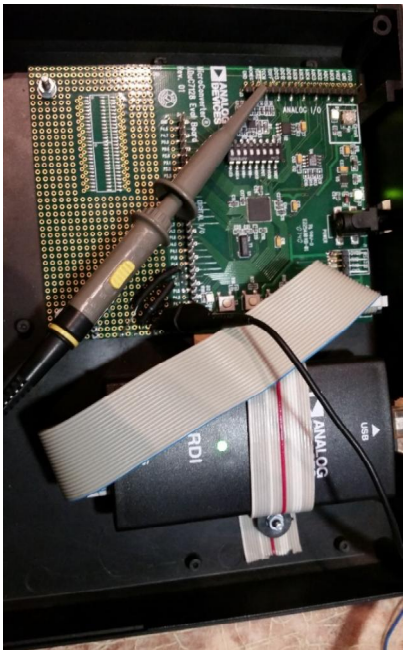


Рис. 6.8. Вікно для відображення покрокового виконання програми



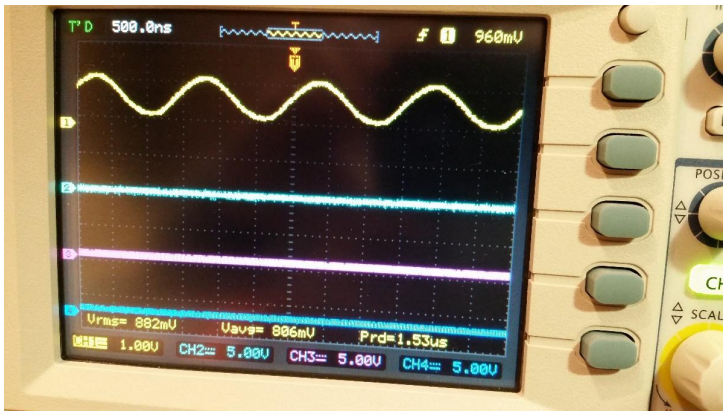


Рис. 6.9. Стенд та відображення виходу DAC на осцилографі

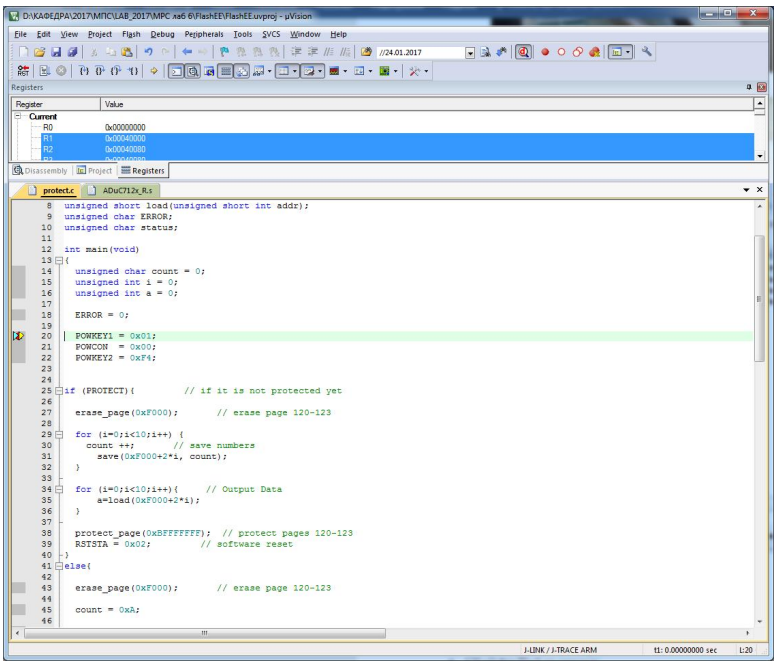


Рис. 6.10. Вікно відлагодження програми

МЕТОДИЧНІ МАТЕРІАЛИ

Мікроконтролер ADuC 7128/7129

Основні характеристики ADuC7128/7129:

- 44 MIPS ARM7TDMI ® ядро;
- 126 кбайти Flash на кристалі;
- 8 кбайт ОЗП (SRAM);
- АЦП, 12 розрядів, 10 каналів, 1 MSPS;
- ЦАП, 10 розрядів;
- DDS (цифровий синтезатор), 32 розряди, 21 МГц;
- прецизійний джерело опорної напруги з малим температурним дрейфом (10 стр / хв / ° С);
- послідовні інтерфейси: 2 × UART, 2 × I2C і SPI;

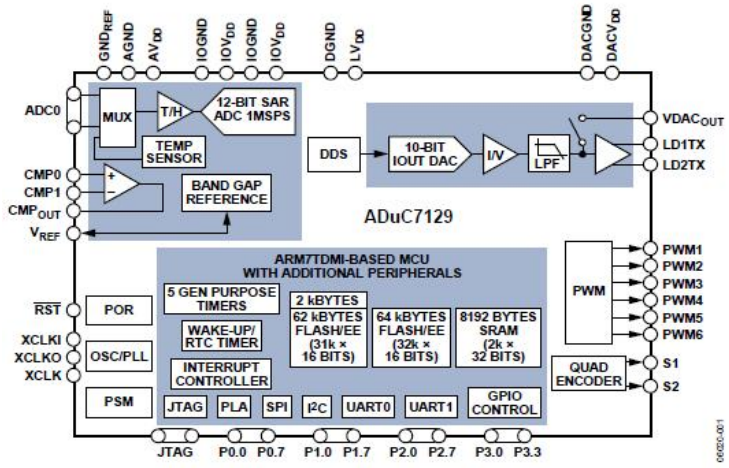


Рис. 6.11. Внутрішня структура мікроконтролера ADuC7128/7129

Характеристика ядра ARM7TDMI

Ядро ARM7TDMI – представник сімейства ARM 32-розрядних мікропроцесорів загального призначення. Сімейство ARM характеризується високою продуктивністю при дуже низькому рівні споживання та малих розмірах.

До областей застосування ядра ARM7 відносять:

- Телекомунікації – контролери GSM терміналів.
- Обмін даними – засоби перетворення протоколів.
- Портативні обчислення – Palmtop комп'ютери.
- Портативні вимірювальні пристрої – кишенькові пристрої збору даних.
- Автомобільну техніку – пристрої керування двигуном.
- Інформаційні системи – Smart карти.
- Засоби відображення – JPEG контролери.

Архітектура ARM виконана на основі принципів RISC (комп'ютер зі скороченим набором інструкцій). Набір інструкцій RISC і пов'язаний з ним механізм дешифрування є більш простим порівняно CISC-архітектурою (комп'ютер зі складним набором інструкцій). За рахунок цього досягається:

- висока продуктивність виконання інструкцій;
- реагування на переривання в режимі реального часу
- малі розміри, ефективна вартість процесорної макрокомірки.

Основні характеристики ядра ARM7

- 32-розрядний RISC процесор (32-розрядні шини даних та адреси).
- 32-розрядна адресація – лінійне адресний простір в 4 Гб – виключає потребу в сегментованої, поділеної на банки або оверлейной пам'яті.
- Тридцять один 32-розрядний регістр загального призначення і шість регістрів стану.
- Регістри адрес, записи й конвеєра.
- Циклічний зсувний пристрій і перемножувач.
- Трирівневий конвеєр (вибірка команди, її декодування і виконання).
- Робочі режими Big Endian і Little Endian.
- Напруга живлення 3,3 і 5 В.
- Мале споживання 0,6 мА при виготовленні по CMOS технології з топологічними нормами 0,8 мкм.
- Повністю статичне робота, що дозволяє додатково знижувати споживання за рахунок зменшення тактової частоти, що ідеально для критичних застосувань.
- Швидкий відгук на переривання, що застосовується для реалізації реального масштабу часу.
- Підтримка систем віртуальної пам'яті.
- Проста але потужна система команд.

Процесор ARM7TDMI підтримує два набори інструкцій: 32-розрядний набір інструкцій ARM; 16-розрядний набір інструкцій Thumb.

Структурна, функціональна схеми процесора і процесорного ядра

На рисунку 1.2 представлені структурна схема процесора ARM7TDMI із зазначенням компонентів і основних сигналів. На рисунку 1.3 демонструється основний процесор (логіка ядра ARM7TDMI), а на рисунку 1.4 показана функціональна схема процесора ARM7TDMI з вказівкою що входять та виходять сигналів.

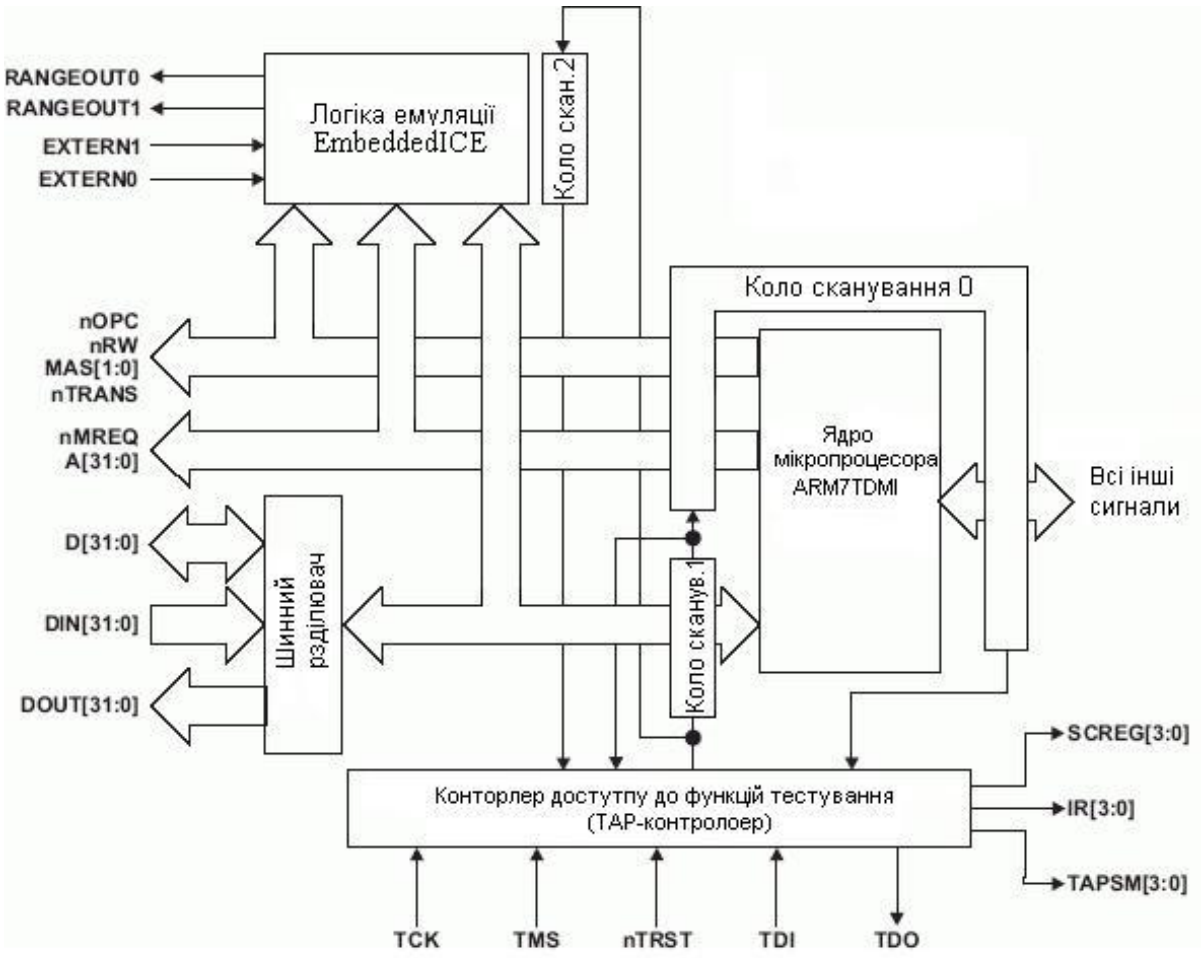


Рис. 6.12. Структурна схема процесора ARM7TDMI

32-розрядна система команд ядра ARM7 містить одинадцять базових типів команд:

- Два типи використовують вбудований арифметико-логічний пристрій, циклічний зсувний пристрій і помножувач при операціях над даними в банку з 31 регістра, форматом по 32 розряду кожен;
- Три класи команд керування переміщенням даних між пам'яттю і регістрами, один оптимізований на забезпечення гнучкості адресації, інший під швидке контекстне перемикання і третій під підкачки даних;
- Три команди керують потоком і рівнем привілегії виконання;
- Три типи призначені для управління зовнішніми сопроцесорами, що дозволяє розширити функціональні можливості системи команд за межами ядра.
- Система команд ARM добре обробляється компіляторами мов високого рівня. На відміну від деяких RISC процесорів, процесор ARM7, при виникненні необхідності в деякому зменшенні обсягу кодів, допускає програмування та на асемблері.

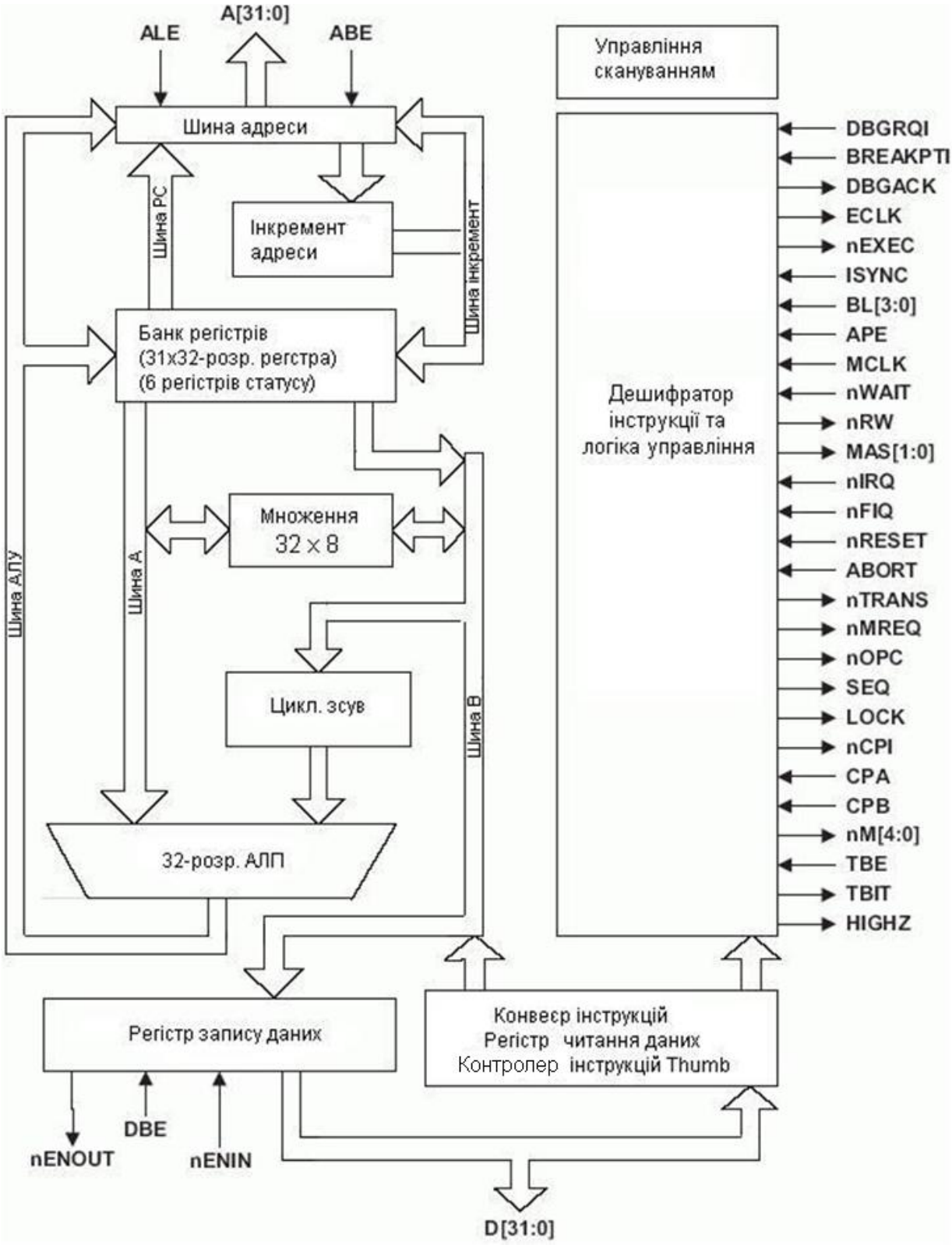


Рис. 6.13. Структурна схема процесорного ядра ARM7TDMI

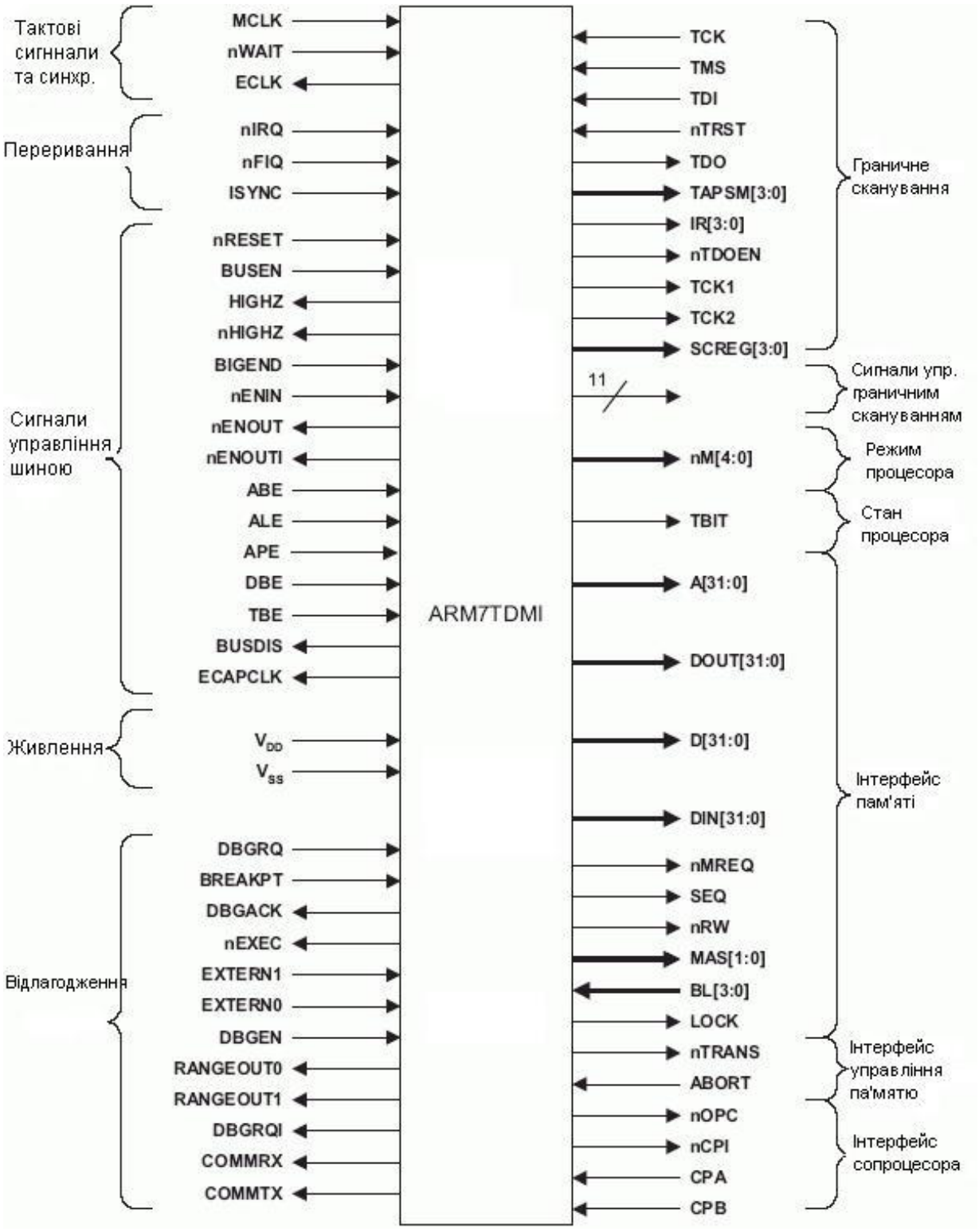


Рис. 6.14. Функціональна схема ядра ARM7TDMI

Організація пам’яті

ADuC7129 об’єднує три окремих блоки пам’яті : 8 kB SRAM і два 64 kB вбудованої Flash/EE пам’яті. 126 kB вбудованої Flash/EE пам’яті доступні користувачу, і тільки 2 kB зарезервовані для фабричних настройок завантажувальної сторінки. Розміщення цих двох блоків показано на рисунку 1.

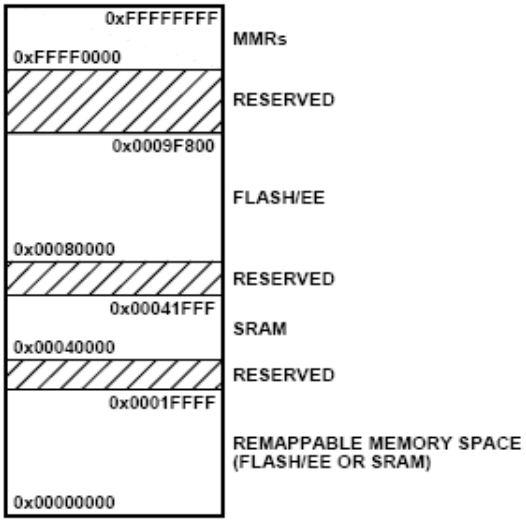


Рис. 6.15. Організація пам’яті ADuC7128/7129

Доступ до пам’яті

ARM7 ядро бачить пам'ять як лінійну таблицю 2^{32} байт, де різні блоки пам’яті розміщені як контури на рисунку 2.

Організація пам’яті ADuC7129 сконфігурована за принципом молодший вперед: найменш значущий байт розміщений в молодшому байті адреси, а найбільш значущий – в старшому байті адреси.

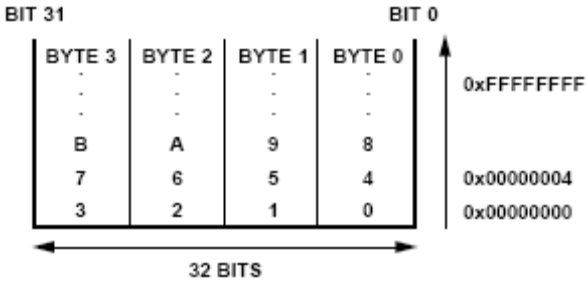


Рис. 6.16. Структура комірки пам’яті

FLASH/EE Пам'ять

128 kB Flash/EE організовано як два блоки 32 k x 16 біт. В першому блоці, 31 k x 16 біт є пам’яттю користувача і 1 k x 16 біт зарезервовані для фабричних настройок завантажувальної сторінки. Розмір сторінки даної Flash/EE пам’яті є 512 байт.

Другий блок 64 kB організовано схожим чином. Він впорядкований як 32 k x 16 біт. Всі вони доступні користувачу.

126 kB Flash/EE є доступними користувачу як пам'ять програм та пам’ять констант. Різниці між пам’ятю даних і пм’ятю програм немає, оскільки ARM програма розподіляє між ними спільний простір. Ширина Flash/EE пам’яті є 16 біт, мається на увазі, що в ARM режимі (32-бітної інструкції), подвійний доступ до Flash/EE є необхідним для витягнення інструкції з пам’яті. Тому для оптимального часу доступу до пам’яті рекомендується використовувати Thumb режим. Максимальна частота роботи Flash/EE пам’яті – 41.78 MHz в Thumb режимі і 20.89 MHz в ARM режимі.

SRAM

8 kB SRAM є доступними користувачу, організовані як 2 k x 32 біти, це є 2 k слів. Оскільки пам'ять SRAM має ширину 32-біт , то ARM програма може запускатись прямо з SRAM на частоті 41.78 MHz .

Набір інструкцій ядра ARM7TDMI.

Набір інструкцій ARM представлений в таблиці 6.1.

Таблиця 6.1. Преставлення інструкцій ARM

Операції		Синтаксис Ассемблера
1	2	3
Пересылка	Пересилання	**MOV** {cond}{S} Rd, <Oprnd2>
	Пересилання NOT	**MVN** {cond}{S} Rd, <Oprnd2>
	Пересилання SPSR в регістр	**MRS** {cond} Rd, SPSR
	Пересилання CPSR в регістр	**MRS** {cond} Rd, CPSR
	Пересилання регістра SPSR	**MSR** {cond} SPSR{field}, Rm
	Пересилання CPSR	**MSR** {cond} CPSR{field}, Rm
	Пересилання константи у прапори SPSR	**MSR** {cond} SPSR_f, #32bit_Imm
	Пересилання константи у прапори CPSR	**MSR** {cond} CPSR_f, #32bit_Imm
Арифметичні	Додавання	**ADD** {cond}{S} Rd, Rn, <Oprnd2>
	Додавання з переносом	**ADC** {cond}{S} Rd, Rn, <Oprnd2>
	Віднімання	**SUB** {cond}{S} Rd, Rn, <Oprnd2>
	Віднімання з переносом	**SBC** {cond}{S} Rd, Rn, <Oprnd2>
	Віднімання зворотнього віднімання	**RSB** {cond}{S} Rd, Rn, <Oprnd2>
	Віднімання зворотнього віднімання з переносом	**RSC** {cond}{S} Rd, Rn, <Oprnd2>
	Множення	**MUL** {cond}{S} Rd, Rm, Rs
	Множення-накопичення	**MLA** {cond}{S} Rd, Rm, Rs, Rn
	Множення длинных беззнаковых чисел	**UMULL** {cond}{S} RdLo, RdHi, Rm, Rs
	Множення – беззнакове накопичення значеннь	**UMLAL** {cond}{S} RdLo, RdHi, Rm, Rs
	Множення знакових даних	**SMULL** {cond}{S} RdLo, RdHi, Rm, Rs
	Множення – знакове накопичення значеннь	**SMLAL** {cond}{S} RdLo, RdHi, Rm, Rs
	Порівняння	**CMP** {cond} Rd, <Oprnd2>
	Порівняння негативне	**CMN** {cond} Rd, <Oprnd2>

Продовження табл. 6.1

1	2	3
Логічні	Перевірка	**TST** {cond} Rn, <Oprnd2>
	Перевірка на еквівалентність	**TEQ** {cond} Rn, <Oprnd2>
	Лог. І	**AND** {cond}{S} Rd, Rn, <Oprnd2>
	Виключне АБО	**EOR** {cond}{S} Rd, Rn, <Oprnd2>
	ORR	**ORR** {cond}{S} Rd, Rn, <Oprnd2>
	Сброс бита	**BIC** {cond}{S} Rd, Rn, <Oprnd2>>
Переход	Перехід	**B** {cond} label
	Перехід за посиланням	**BL** {cond} label
	Перехід і зміні набору інструкцій	**BX** {cond} Rn
Чтение	Читання слова	**LDR** {cond} Rd, <a_mode2>
	Читання слова з перевагою режиму користувача	**LDR** {cond}T Rd, <a_mode2P>
	Читання байта	**LDR** {cond}B Rd, <a_mode2>
	Читання байта з перевагою режиму користувача	**LDR** {cond}BT Rd, <a_mode2P>
	Читання байта зі знаком	**LDR** {cond}SB Rd, <a_mode3>
	Читання напівслова	**LDR** {cond}H Rd, <a_mode3>
	Читання напівслова зі знаком	**LDR** {cond}SH Rd, <a_mode3>
	Читання операции с декількома блоками даних	-
	– з попереднім інкрементом	**LDM** {cond}IB Rd{!}, <reglist>{^}
	– з подальшим інкрементом	**LDM** {cond}IA Rd{!}, <reglist>{^}
	– с предварительным декрементом	**LDM** {cond}DB Rd{!}, <reglist>{^}
	– з попереднім декрементом	**LDM** {cond}DA Rd{!}, <reglist>{^}
	– операція над стеком	**LDM** {cond}<a_mode4L> Rd{!}, <reglist>
	– операція над стеком і відновлення CPSR	**LDM** {cond}<a_mode4L> Rd{!}, <reglist+pc>^
	операція над стеком з регістрами користувача	**LDM** {cond}<a_mode4L> Rd{!}, <reglist>^
Запис	Запис слова	**STR** {cond} Rd, <a_mode2>
	Запис слова з перевагою режиму користувача	**STR** {cond}T Rd, <a_mode2P>
	Запис байта	**STR** {cond}B Rd, <a_mode2>
	Запис байта з перевагою режиму користувача	**STR** {cond}BT Rd, <a_mode2P>
	Запис полуслова	**STR** {cond}H Rd, <a_mode3>
	Запис операції над декількома блоками даних	-
	– з попереднім інкрементом	**STM** {cond}IB Rd{!}, <reglist>{^}
	– з наступним інкрементом	**STM** {cond}IA Rd{!}, <reglist>{^}
	– з попереднім декрементом	**STM** {cond}DB Rd{!}, <reglist>{^}
	– з наступним декрементом	**STM** {cond}DA Rd{!}, <reglist>{^}
	– операція над стеком	**STM** {cond}<a_mode4S> Rd{!}, <reglist>
	– операція над стеком з регістрами користувача	**STM** {cond}<a_mode4S> Rd{!}, <reglist>^

Закінчення табл. 6.1

1	2	3
Обмін	слів	**SWP** {cond} Rd, Rm, [Rn]
	байт	**SWP** {cond}B Rd, Rm, [Rn]
Сопроцессор	Операція над байтами	**CDP** {cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>
	Пересилання в ARM-регістр з сопроцесора	**MRC** {cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>
	Пересилання в сопроцесор з ARM-регістра	**MCR** {cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>
	Читання	**LDC** {cond} p<cpnum>, CRd, <a_mode5>
	Запис	**STC** {cond} p<cpnum>, CRd, <a_mode5>
Програмне переривання		**SWI** 24bit_Imm

Стиснення інструкцій

Мікропроцесорні архітектури, як правило, використовують набір інструкцій тієї ж розрядності, що і дані. Отже, 32-розрядні архітектури володіють поліпшеними характеристиками обробки 32-розрядних даних і можуть більш ефективно адресувати великі адресні простору в порівнянні з 16-розрядними архітектурою. 16-розрядні архітектури в більшості випадків мають більш високою щільністю коду, але приблизно вдвічі поступаються за продуктивністю.Thumb реалізує 16-розрядний набір інструкцій у складі 32-розрядної архітектури, який забезпечує:

- більш високу продуктивність у порівнянні з 16-розрядної архітектурою
- більш високу щільність коду в порівнянні з 32-розрядної архітектурою.

Набір інструкцій Thumb

Набір інструкцій Thumb є вибіркою найбільш часто використовуваних 32-розрядних інструкцій ARM. Thumb-інструкції характеризуються розміром 16 біт та мають відповідну 32-розрядну інструкцію ARM. Відповідні інструкції надають однаковий вплив на модель процесора. Thumb-інструкції працюють зі стандартною конфігурацією ARM-регістру, забезпечуючи чудову взаємодія між станами ARM і Thumb.

У процесі виконання 16-розрядна інструкція піддається декомпресії в реальному часі до повних 32-розрядних інструкцій ARM без втрати продуктивності.

Thumb успадковує всі переваги 32-розрядного ядра:

- 32-розрядне адресний простір
- 32-розрядні регістри
- 32-розрядне зсувне пристрій і арифметико-логічного пристрій (АЛП)
- Пересилання в пам'ять 32-розрядних даних.

Отже, інструкції Thumb характеризуються великим діапазоном переходу, потужними арифметичними операціями і великим простором. Код Thumb зазвичай займає 65% від ARM-коду і досягає 160% продуктивності ARM-коду при виконанні з 16-розрядної системи пам'яті. Отже, Thumb, робить ядро ARM7TDMI ідеально підходящим під вбудовані додатки, де в якості критичних параметрів виступають щільність коду і габарити. Доступність обох наборів інструкцій, 16-розрядного Thumb і 32-розрядного ARM, дає розробникам гнучкість по оптимізації швидкодії або розміру коду на рівні процедури

відповідно до вимог до їх програми. Наприклад, критичні цикли у таких програмах, як обробка даних, алгоритми цифрової обробки сигналів, часто виникають переривання, які можуть кодуватися за допомогою повних ARM-інструкцій, а потім бути перетворені в Thumb-код.

Конвеєр інструкцій

Для прискорення потоку інструкцій, що надходить до процесора, в ядрі ARM7TDMI використовується конвеєр. Він дозволяє виконувати кілька операцій одночасно, а також забезпечує паралельну роботу системи пам'яті і вузлів обробки. Використовується триступінчата конвейеризації, тому що інструкції виконуються в три етапи:

- Вибірка
- Дешифрування
- Виконання.

Конвеєр інструкцій показаний на рисунку.

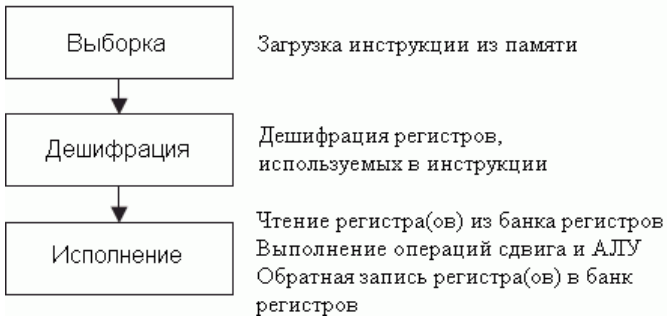


Рис. 6.17. Конвеєр інструкцій

У процесі нормальної роботи, коли виконується одна інструкція, наступна інструкція декодується, а третій – зчитується з пам'яті. Лічильник програми вказує на що завантажується з пам'яті інструкцію, а не на виконувану інструкцію. Це важливо знати, тому що значення лічильника програми (PC), що використовується при виконанні інструкції, завжди на дві інструкції випереджає адресу.

Доступ до пам'яті

Ядро ARM7TDMI використовує фон-неймановскую архітектуру доступу до пам'яті з одною 32-розрядної шиною даних, по якій передаються як дії, так і дані. Доступ до даних до пам'яті можуть здійснювати тільки інструкції читання, запису та обміну. В якості

даних можуть виступати:

8 біт (байт);

16 біт (півслова);

32 біт (слово).

Слова повинні бути вирівняні до 4-байтним кордонів, а півслова – до 2-байтним.

Інтерфейс пам'яті

При розробці інтерфейсу пам'яті процесора ARM7TDMI стояло завдання максимально повністю використовувати потенціал за швидкодією і при цьому мінімізувати використання пам'яті. Щоб забезпечити можливість реалізації функцій системного управління на основі стандартної малопотужною логіки критичні до швидкодії сигнали управління були конвейерізовани. Дані сигнали управління спрощують використання швидкодіючих

пакетних режимів передачі, які використовуються більшістю вбудованої і зовнішньої технологій пам'яті.

Ядро ARM7TDMI підтримує чотири основних цикли доступу до пам'яті:

холостий цикл

непослідовний цикл

послідовний цикл

цикл передачі регістра співпроцесора.

Логіка EmbeddedICE

Логіка EmbeddedICE – допоміжний апаратний блок, який робить ARM-процесори налагоджували і істотно полегшує процес налагодження. Він дозволяє за допомогою програмних інструментальних засобів налагоджувати програмний код, який виконується цільовим процесором. Логіка EmbeddedICE управляється через порт JTAG з використанням інтерфейсу EmbeddedICE.

Режими адресації

Режими адресації – процедури, які використовуються різними інструкціями для генерації значень, що використовуються інструкціями. Процесор ARM7TDMI підтримує 5 режимів адресації:

Режим 1 – зсувне операнди інструкцій для обробки даних.

Режим 2 – Читання і запис слова або беззнакового байта.

Режим 3 – Читання і запис півслова або завантаження знакового байта.

Режим 4 – Множинні читання і запис.

Режим 5 – Читання і запис співпроцесора.

Режими роботи Процесор ARM7TDMI підтримує сім режимів роботи:

1. Режим користувача – звичайний стан ARM при виконанні програми, також використовується для виконання більшості прикладних програм.

2. Режим швидкого переривання (FIQ), який підтримує передачу даних або обробку каналу.

3. Режим переривання (IRQ), який використовується для обробки переривань загального призначення.

4. Супервізорний режим, який є захищеним режимом для операційної системи.

5. Аварійний режим, який вводиться після аварійної вибірки даних або інструкції.

6. Системний режим – привілейований режим користувача для операційної системи.

Прим.: Ви можете вводити системний режим з іншого привілейованого режиму тільки шляхом зміни біта режиму в регістрі поточного стану програми (CPSR). 7. Невизначений режим вводиться, коли виконується невизначена інструкція. Всі режими (крім режиму користувача) називаються привілейованими режимами. Привілейовані режими використовуються для обслуговування переривань і виняткових ситуацій, а також для доступу до захищених ресурсів.

Регістри

Процесор ARM7TDMI містить усього 37 регістрів:

- 31 32-розрядних регістра загального призначення
- 6 регістрів статусу.

Не всі регістри доступні в один і той же час. Доступність регістрів для програміста залежить від стану процесора і робочого режиму.

Набір регістрів в режимі ARM

У режимі ARM доступні 16 регістрів загального призначення, один або два регістра статусу.У привілейованих режимах стають доступними специфічні банки регістрів. На рис.

2.3 демонструється, які регістри доступні в кожному режимі. У набір регістрів в режимі ARM входять 16 регістрів r0 ... r15. Ще один регістр, CPSR, містить прапори умови коду та біти поточного режиму. Регістри r0 ... r13 є регістрами загального призначення і можуть використовуватися як для зберігання даних, так і для зберігання адреси.

Регістри r14 та r15 виконують наступні спеціальні функції:

Регістр зв'язку

Регістр 14 використовується як регістр зв'язку (LR) підпрограми. Регістр r14 приймає копію регістра r15 при виконанні інструкції перехід по посиланню (BL). У всіх інших випадках регістр r14 може використовуватися як регістр загального призначення. Відповідні банкірованние регістри r14_svc, r14_irq, r14_fiq, r14_abt і r14_und аналогічним чином використовуються для запам'ятовування значень повернення r15 при виникненні переривань і виняткових ситуацій або при виконанні інструкції BL усередині процедур обробки переривань або виняткових ситуацій.

Лічильник програми

Регістр 15 зберігає значення PC. У режимі ARM біти [1:0] регістра r15 мають невизначений значення і повинні ігноруватися. Біти [31:2] містять значення PC.

У стані Thumb біт [0] має невизначене значення і повинен ігноруватися. Біти [31:1] містять значення PC. Регістр r13 використовується як показник стека (SP).

У привілейованих режимах доступний ще один регістр – регістр зберігання статусу програми (SPSR). Він містить прапори коду умови й біти режиму, що є результатом виняткової ситуації, яке викликало входження в поточний режим. Банки регістрів – дискретні фізичні регістри в ядрі, які знаходяться в позиціях доступних регістрів в залежності від поточного робочого режиму процесора. Вміст банкірованного регістра запам'ятовується при змінах робочих режимів. У режимі FIQ є сім банкірованних регістрів в позиціях r8-r14 (r8_fiq-r14_fiq). У стані ARM кілька обробників швидких переривань (FIQ) не повинні виконувати запис в будь-якій регістр.

У режимах користувача, IRQ, супервізорном, аварійному та невизначеному є два банкірованних регістра в позиції r13 та r14, дозволяючи зберігати власне значення SP і LR в кожному режимі.

У системному режимі використовуються ті ж регістри, що і в режимі користувача.

Рис. 6.18. Організація регістрів в режимі ARM

Цифро-аналоговий перетворювач (DAC) та цифровий синтезатор частоти(DDS)

Мікроконтролери ADuC7128/ADuC7129 мають вбудований 10-бітний цифро-аналоговий перетворювач (DAC), який може використовуватися для формування заданих користувачем сигналів різної форми або синусоїдальних сигналів, що генеруються цифровим синтезатором частоти (DDS). Сигнал DAC формується з 10-бітним струмовим IDAC з подальшим перетворенням струму в напругу. На виході з IDAC є RC-фільтр, який перетворює струм в напругу. Ця напруга поступає на операційний підсилювач (ОП) з якого поступає на вихідну лінію. Для функціонування DAC використовується внутрішнє джерело опорної напруги 2,5В, яке включається встановленням REFCON=0x01.

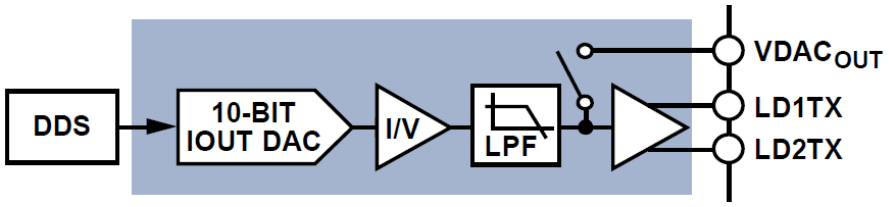
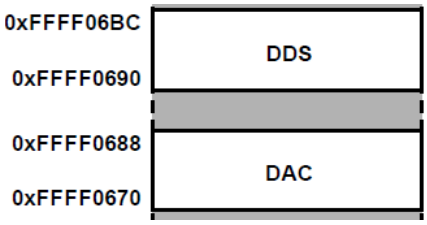


Рис. 6.19. Внутрішня структура DAC



Address	Name	Byte	Access Type
0x0670	DACCON	2	R/W
0x0690	DDSCON	1	R/W
0x0694	DDSFRQ	4	R/W
0x0698	DDSPHS	2	R/W
0x06A4	DACKEY0	1	R/W
0x06B4	DACDAT	2	R/W
0x06B8	DACEN	1	R/W
0x06BC	DACKEY1	1	R/W

Рис. 6.20. Регістри DDS та DAC

Регістри DAC:

Таблиця 6.2. Структура регістра **DACCON**

Bit	Value	Description
10:9		Reserved. These bits should be written to 0 by the user.
8		Reserved. This bit should be written to 0 by the user.
7		Reserved. This bit should be written to 0 by the user.
6		Reserved. This bit should be written to 0 by the user.
5		Output Enable. This bit operates in all modes. In Line Driver mode, this bit should be set. Set by user to enable the line driver output. Cleared by user to disable the line driver output. In this mode the line driver output is high impedance.
4		Single-Ended or Differential Output Control. Set by user to operate in differential mode, the output is the differential voltage between LD1TX and LD2TX. The voltage output range is $V_{REF}/2 \pm V_{REF}/2$. Cleared by user to reference the LD1TX output to AGND. The voltage output range is $AV_{DD}/2 \pm V_{REF}/2$.
3		Reserved. This bit should be set to 0 by the user.
2:1		Operation Mode Control. This bit selects the mode of operation of the DAC.
	00	Power-Down.
	01	Reserved.
	10	Reserved.
	11	DDS and DAC Mode. Selected by DACEN.
0		DAC Update Rate Control. This bit has no effect when in DDS mode. Set by user to update the DAC on the negative edge of Timer1. This allows the user to use any one of the core CLK, OSC CLK, baud CLK, or user CLK and divide these down by 1, 16, 256, or 32,768. A user can do waveform generation by writing to the DAC data register from RAM and updating the DAC at regular intervals via Timer1. Cleared by user to update the DAC on the negative edge of HCLK.

Name	Address	Default Value	Access
DACEN	0xFFFF06B8	0x00	R/W

Bit	Description
7:1	Reserved.
0	Set to 1 by the user to enable DAC mode. Set to 0 by the user to enable DDS mode.

Name	Address	Default Value	Access
DACDAT	0xFFFF06B4	0x0000	R/W

Bit	Description
15:10	Reserved.
9:0	10-bit data for DAC.

DACEN	DACDAT
DACKEY0 = 0x07	DACKEY0 = 0x07
DACEN = user value	DACDAT = user value
DACKEY1 = 0xB9	DACKEY1 = 0xB9

Регістри DDS:

Таблиця 6.3. Структура регістра **DDSCON**

Bit	Description	
7:6	Reserved.	
5	DDS Output Enable. Set by user to enable the DDS output. This has an effect only if the DDS is selected in DACCON. Cleared by user to disable the DDS output.	
4	Reserved.	
3:0	Binary Divide Control.	
	DIV	**Scale Ratio**
	0000	0.000
	0001	0.125
	0010	0.250
	0011	0.375
	0100	0.500
	0101	0.625
	0110	0.750
	0111	0.875
	1xxx	1.000

Name	Address	Default Value	Access
DDSFRQ	0xFFFF0694	0x00000000	R/W

Bit	Description
31:0	Frequency select word (FSW)

$$Frequency = \frac{FSW \times 20.8896\ \text{MHz}}{2^{32}}$$

Name	Address	Default Value	Access
DDSPHS	0xFFFF0698	0x00000000	R/W

Bit	Description
31:12	Reserved
11:0	Phase

$$Phase\ Offset = \frac{2 \times \pi \times Phase}{2^{12}}$$

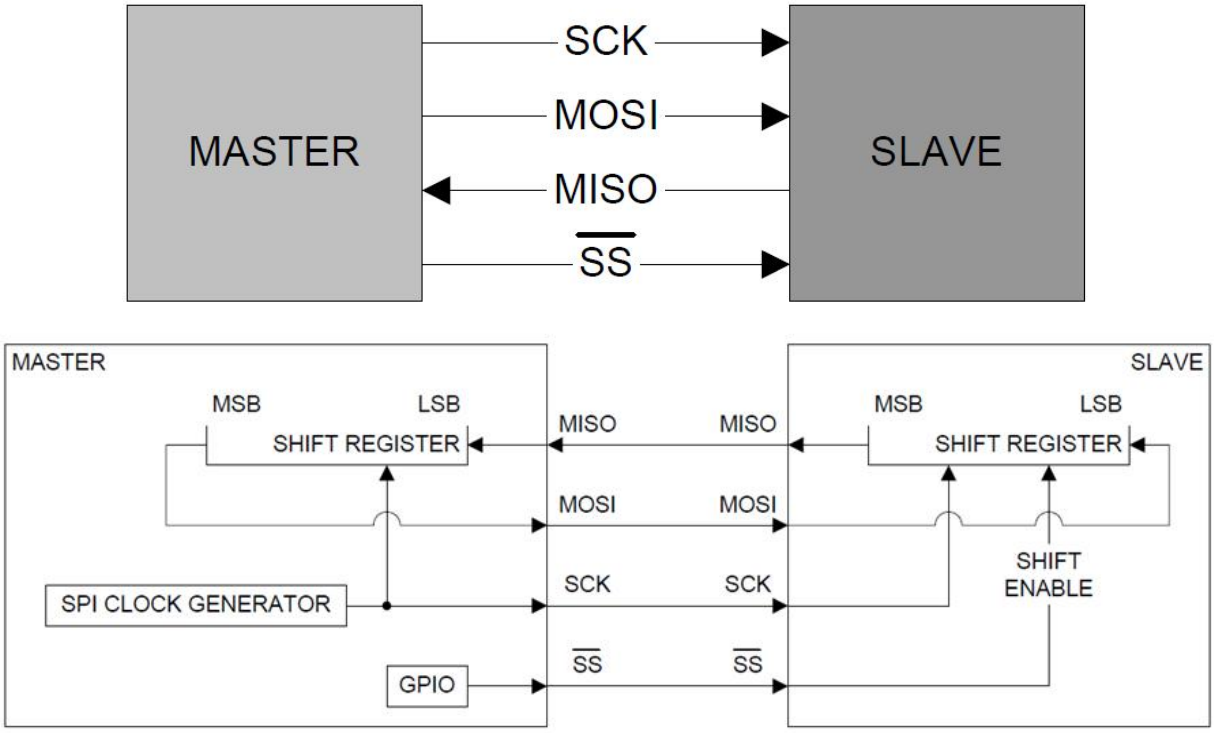
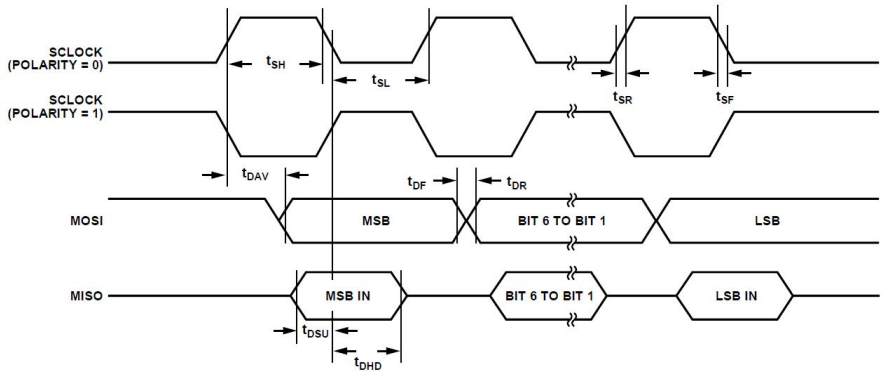


Рис. 6.21. Структура вузла SPI

SPI Master Mode Timing (PHASE Mode = 1)

Parameter	Description	Min	Typ	Max	Unit
t_{SL}	SCLOCK low pulse width[1]		(SPIDIV + 1) × t_{HCLK}		ns
t_{SH}	SCLOCK high pulse width[1]		(SPIDIV + 1) × t_{HCLK}		ns
t_{DAV}	Data output valid after SCLOCK edge			2 × t_{HCLK} + 2 × t_{UCLK}	ns
t_{DSU}	Data input setup time before SCLOCK edge[2]	1 × t_{UCLK}			ns
t_{DHD}	Data input hold time after SCLOCK edge[2]	2 × t_{UCLK}			ns
t_{DF}	Data output fall time		5	12.5	ns
t_{DR}	Data output rise time		5	12.5	ns
t_{SR}	SCLOCK rise time		5	12.5	ns
t_{SF}	SCLOCK fall time		5	12.5	ns

[1] t_{HCLK} depends on the clock divider or CD bits in the PLLCON MMR, $t_{HCLK} = t_{UCLK}/2^{CD}$.
[2] t_{UCLK} = 23.9 ns. It corresponds to the 41.78 MHz internal clock from the PLL before the clock divider.



SPI Master Mode Timing (PHASE Mode = 0)

Parameter	Description	Min	Typ	Max	Unit
t_{SL}	SCLOCK low pulse width[1]		(SPIDIV + 1) × t_{HCLK}		ns
t_{SH}	SCLOCK high pulse width[1]		(SPIDIV + 1) × t_{HCLK}		ns
t_{DAV}	Data output valid after SCLOCK edge			2 × t_{HCLK} + 2 × t_{UCLK}	ns
t_{DOSU}	Data output setup before SCLOCK edge			75	ns
t_{DSU}	Data input setup time before SCLOCK edge[2]	1 × t_{UCLK}			ns
t_{DHD}	Data input hold time after SCLOCK edge[2]	2 × t_{UCLK}			ns
t_{DF}	Data output fall time		5	12.5	ns
t_{DR}	Data output rise time		5	12.5	ns
t_{SR}	SCLOCK rise time		5	12.5	ns
t_{SF}	SCLOCK fall time		5	12.5	ns

[1] t_{HCLK} depends on the clock divider or CD bits in the PLLCON MMR, $t_{HCLK} = t_{UCLK}/2^{CD}$.
[2] t_{UCLK} = 23.9 ns. It corresponds to the 41.78 MHz internal clock from the PLL before the clock divider.

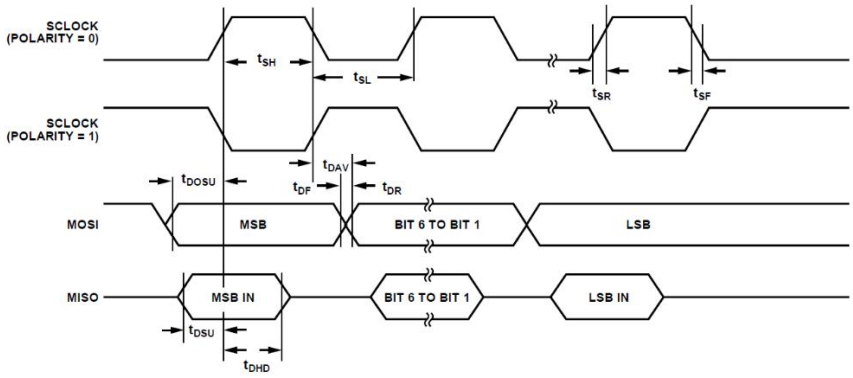
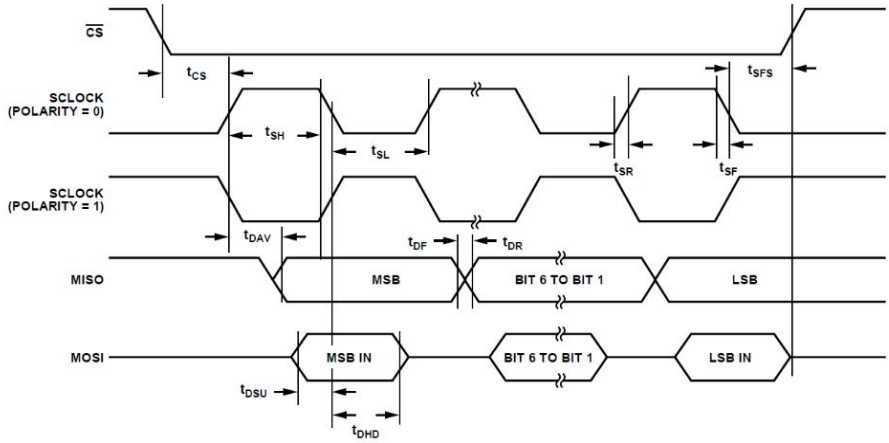


*Рис. 6.22. Часові діаграми в режимі **Master***

SPI Slave Mode Timing (PHASE Mode = 1)

Parameter	Description	Min	Typ	Max	Unit
t_{CS}	CS to SCLOCK edge[1]	2 × t_{UCLK}			ns
t_{SL}	SCLOCK low pulse width[2]		(SPIDIV + 1) × t_{HCLK}		ns
t_{SH}	SCLOCK high pulse width[2]		(SPIDIV + 1) × t_{HCLK}		ns
t_{DAV}	Data output valid after SCLOCK edge			2 × t_{HCLK} + 2 × t_{UCLK}	ns
t_{DSU}	Data input setup time before SCLOCK edge[1]	1 × t_{UCLK}			ns
t_{DHD}	Data input hold time after SCLOCK edge[1]	2 × t_{UCLK}			ns
t_{DF}	Data output fall time		5	12.5	ns
t_{DR}	Data output rise time		5	12.5	ns
t_{SR}	SCLOCK rise time		5	12.5	ns
t_{SF}	SCLOCK fall time		5	12.5	ns
t_{SFS}	CS high after SCLOCK edge	0			ns

[1] t_{UCLK} = 23.9 ns. It corresponds to the 41.78 MHz internal clock from the PLL before the clock divider.
[2] t_{HCLK} depends on the clock divider or CD bits in the PLLCON MMR, $t_{HCLK} = t_{UCLK}/2^{CD}$.



SPI Slave Mode Timing (PHASE Mode = 0)

Parameter	Description	Min	Typ	Max	Unit
t_{CS}	CS to SCLOCK edge[1]	2 × t_{UCLK}			ns
t_{SL}	SCLOCK low pulse width[2]		(SPIDIV + 1) × t_{HCLK}		ns
t_{SH}	SCLOCK high pulse width[2]		(SPIDIV + 1) × t_{HCLK}		ns
t_{DAV}	Data output valid after SCLOCK edge			2 × t_{HCLK} + 2 × t_{UCLK}	ns
t_{DSU}	Data input setup time before SCLOCK edge[1]	1 × t_{UCLK}			ns
t_{DHD}	Data input hold time after SCLOCK edge[1]	2 × t_{UCLK}			ns
t_{DF}	Data output fall time		5	12.5	ns
t_{DR}	Data output rise time		5	12.5	ns
t_{SR}	SCLOCK rise time		5	12.5	ns
t_{SF}	SCLOCK fall time		5	12.5	ns
t_{DOCS}	Data output valid after CS edge			25	ns
t_{SFS}	CS high after SCLOCK edge	0			ns

[1] t_{UCLK} = 23.9 ns. It corresponds to the 41.78 MHz internal clock from the PLL before the clock divider.
[2] t_{HCLK} depends on the clock divider or CD bits in the PLLCON MMR, $t_{HCLK} = t_{UCLK}/2^{CD}$.

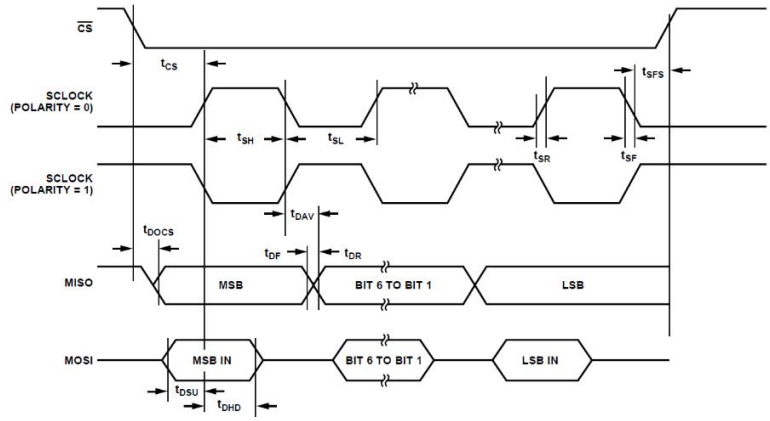


*Рис. 6.23. Часові діаграми в режимі **Slave***

Таблиця 6.4. Мультиплексований порт SPM – SERIAL PORT MUX

SPM4	P1.4	RI0	SPICLK	PLAI[4]
SPM5	P1.5	DCD0	SPIMISO	PLAI[5]
SPM6	P1.6	DSR0	SPIMOSI	PLAI[6]
SPM7	P1.7	DTR0	SPICSL	PLAO[0]

SPICLK	(Serial Clock)	I/O Pin
SPIMISO	(Master In, Slave Out)	I/O Pin
SPIMOSI	(Master Out, Slave In)	I/O Pin
SPICSL	Chip Select (CS)	Input Pin

Таблиця 6.5а. Регістри SPI

SPI Speed vs. Clock Divider Bits in Master Mode

CD Bits	0	1	2	3	4	5
SPIDIV in hex	0x05	0x0B	0x17	0x2F	0x5F	0xBF
SPI speed in MHz	3.482	1.741	0.870	0.435	0.218	0.109

$$f_{SERIAL\ CLOCK} = \frac{f_{HCLK}}{2 \times (1 + SPIDIV)}$$

Таблиця 6.5б. Регістри SPI

SPISTA Register

Name	Address	Default Value	Access
SPISTA	0xFFFF0A00	0x00	R

SPISTA is an 8-bit read-only status register.

SPISTA MMR Bit Designations

Bit	Description
7:6	Reserved.
5	SPIRX Data Register Overflow Status Bit. Set if SPIRX is overflowing. Cleared by reading SPIRX register.
4	SPIRX Data Register IRQ. Set automatically if Bit 3 or Bit 5 is set. Cleared by reading SPIRX register.
3	SPIRX Data Register Full Status Bit. Set automatically if valid data is present in the SPIRX register. Cleared by reading SPIRX register.
2	SPITX Data Register Underflow Status Bit. Set automatically if SPITX is underflowing. Cleared by writing in the SPITX register.
1	SPITX Data Register IRQ. Set automatically if Bit 0 is clear or Bit 2 is set. Cleared by writing in the SPITX register or if finished transmission disabling the SPI.
0	SPITX Data Register Empty Status Bit. Set by writing to SPITX to send data. This bit is set during transmission of data. Cleared when SPITX is empty.

Таблиця 6.5в. Регістри SPI

SPIRX Register

Name	Address	Default Value	Access
SPIRX	0xFFFF0A04	0x00	R

SPIRX is an 8-bit read-only receive register.

SPITX Register

Name	Address	Default Value	Access
SPITX	0xFFFF0A08	0x00	W

SPITX is an 8-bit write-only transmit register.

SPIDIV Register

Name	Address	Default Value	Access
SPIDIV	0xFFFF0A0C	0x1B	R/W

SPIDIV is an 8-bit serial clock divider register.

SPICON Register

Name	Address	Default Value	Access
SPICON	0xFFFF0A10	0x0000	R/W

SPICON is a 16-bit control register.

SPICON MMR Bit Designations

Bit	Description
15:13	Reserved.
12	Continuous Transfer Enable. Set by user to enable continuous transfer. In master mode, the transfer continues until no valid data is available in the TX register. $\overline{CS}$ is asserted and remains asserted for the duration of each 8-bit serial transfer until TX is empty. Cleared by user to disable continuous transfer. Each transfer consists of a single 8-bit serial transfer. If valid data exists in the SPITX register, then a new transfer is initiated after a stall period.
11	Loopback Enable. Set by user to connect MISO to MOSI and test software. Cleared by user to be in normal mode.
10	Slave Output Enable. Set by user to enable the slave output. Cleared by user to disable slave output.
9	Slave Select Input Enable. Set by user in master mode to enable the output.
8	SPIRX Overflow Overwrite Enable. Set by user, the valid data in the RX register is overwritten by the new serial byte received. Cleared by user, the new serial byte received is discarded.
7	SPITX Underflow Mode. Set by user to transmit 0. Cleared by user to transmit the previous data.
6	Transfer and Interrupt Mode (Master Mode). Set by user to initiate transfer with a write to the SPITX register. Interrupt occurs when TX is empty. Cleared by user to initiate transfer with a read of the SPIRX register. Interrupt occurs when RX is full.
5	LSB First Transfer Enable Bit. Set by user, the LSB is transmitted first. Cleared by user, the MSB is transmitted first.
4	Reserved. Should be set to 0.
3	Serial Clock Polarity Mode Bit. Set by user, the serial clock idles high. Cleared by user, the serial clock idles low.
2	Serial Clock Phase Mode Bit. Set by user, the serial clock pulses at the beginning of each serial bit transfer. Cleared by user, the serial clock pulses at the end of each serial bit transfer.
1	Master Mode Enable Bit. Set by user to enable master mode. Cleared by user to enable slave mode.
0	SPI Enable Bit. Set by user to enable the SPI. Cleared to disable the SPI.

Лабораторний стенд на базі EVAL-ADUC7128QSPZ

Рис. 6.24. Комплектація EVAL-ADUC7128QSPZ

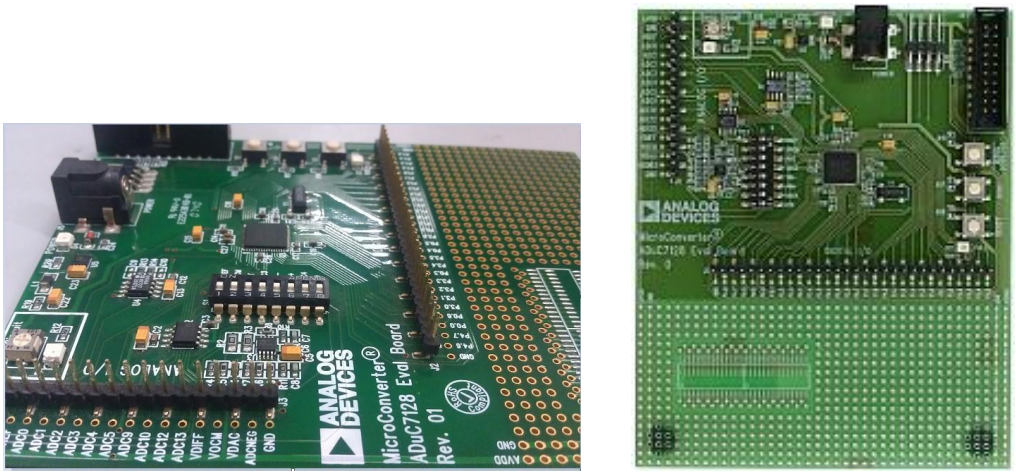


Рис. 6.25. Плата EVAL-ADUC7128QSPZ

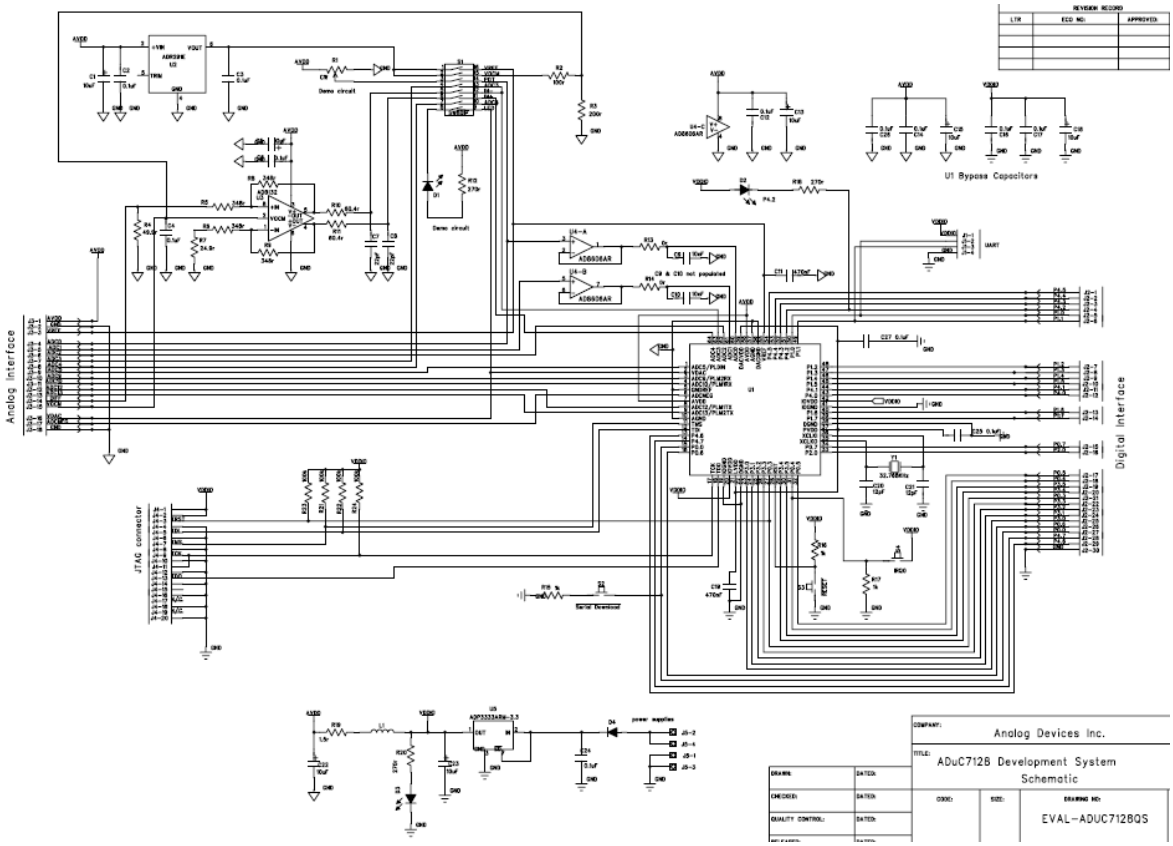


Рис. 6.26. Схема плати EVAL-ADUC7128QSPZ

Література

6.1. ADUC7128_7129.pdf.
6.2. ADuC7128_ds.pdf.
6.3. ADuC7XXXGetStartedGuideV0 4.pdf
6.4. Конспект лекцій з дисципліни “Мікропроцесорні системи”

Лістинг програми

```
/*****************************************************************************
 File           : DACsine.c

 Hardware       : ADuC7128 Rev C silicon

 Description    : Uses the DDS to output a sine wave on the DAC

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

#include "ADuC7128.h"
void delay (int length);
int main (void)  {
        GP4DAT = 0x04000000;    // P4.2 configured as an output. LED is turned on

        PLLKEY1 = 0xAA;         // switch to external crystal
        PLLCON  = 0x01;
        PLLKEY2 = 0x55;

        REFCON = 0x01;          // Reference must be decoupled for DAC operation

        DACCON = 0x3E;          // DAC set in DDS mode
        DDSCON = 0x28;          // DDS output enabled
        DDSFRQ = 0x08000000;    // Freq set to 652kHz

        while(1){
        GP4DAT ^= 0x00040000;   // Complement P4.2
        delay(100000);
        }


}

void delay (int length)
{
        while (length >=0)
        length--;
}


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

  File          : master.c

  Hardware      : Applicable to ADuC7128 rev b or c silicon

  Description   : SPI master     to demonstrate with slave.c or slave1.c

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


 #include<ADuC7128.h>

 unsigned char results[30] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
        0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,
        0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x20,
        0x21, 0x22, 0x23, 0x24};

 int main(void) {
        int i = 0;

        GP1CON = 0x22220000;    // configure SPI on SPM
        SPIDIV = 0xCC;          // set SPI clock 40960000/(2x(1+SPIDIV))
                                // 0xCC = 100kHz
        SPICON = 0x104B;        // enable SPI master in continuous transfer mode
                                // slave select will stay low during the all transmission
        for (i=0;i<30;i++)
        {
                SPITX = results[i];  // transmit command or any dummy data
                while ((SPISTA & 0x02) != 0x02) ; // wait for data received status bit
        }
 while (1){}
 }
```

```
/*****************************************************************************

 File           : slave.c

 Hardware       : Applicable to ADuC7128 rev b or c silicon
                  Currently targetting ADuC7128.

 Description    : SPI slave is to use with master.c or master1.c
                  the slaves receives values from the master and
                  keeps transmitting '0' as it is the default value at reset.

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


#include<ADuC7128.h>


int main(void) {
char i;
char received_data[30];

        GP1CON = 0x22220000;            // configure SPI on SPM
        SPICON = 0x1409;                // enable SPI slave mode


for (i=0; i <30; i++) {
        while (!(SPISTA & 0x08)) ;      // wait for data in the RX MMR
        received_data[i] = SPIRX;       // read data and clear bit 4 of SPISTA
        }
while (1) {

        }
```

ЗМІСТ

ЕЛЕКТРОННЕ НАВЧАЛЬНЕ ВИДАННЯ

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
МІКРОПРОЦЕСОРНІ СИСТЕМИ

Навчальний посібник

Комп'ютерне верстання *Наталії Максимюк*
Коректор *Софія Голько*

Режим доступу: http://eom.lp.edu.ua/textbooks/np-mps.pdf

Видавець і виготівник: Видавництво Львівської політехніки
Свідоцтво суб'єкта видавничої справ и ДК № 4459 від 27.12.2012 р.

вул. Ф. Колесси, 4, Львів, 79013
тел. +380 32 2584103, факс +380 32 2584101
vlp.com.ua, ел. пошта: vmr@vlp.com.ua