Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ
КАФЕДРА «ПРОМИСЛОВОЇ ЕЛЕКТРОНІКИ»

РОЗПОДІЛЕНІ МІКРОПРОЦЕСОРНІ СИСТЕМИ

ПРАКТИЧНІ ЗАНЯТТЯ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для аспірантів,
які навчаються за спеціальністю 171 «Електроніка»,
спеціалізацією «Електронні компоненти і системи»

Київ 2018

Розподілені мікропроцесорні системи: практичні заняття [Електронний ресурс]: для підготовки докторів філософії в галузі знань 17 Електроніка та телекомунікація за спеціальністю 171 Електроніка за спеціалізацією «Електронні системи» / КПІ ім. Ігоря Сікорського ; уклад.: Т. О. Терещенко – Електронні текстові данні (1 файл: 4029 кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 50 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № від р.)
за поданням Вченої ради факультету електроніки (протокол № від .2018 р.)

Електронне мережне навчальне видання

РОЗПОДІЛЕНІ МІКРОПРОЦЕСОРНІ СИСТЕМИ

Практичні заняття для аспірантів напрямку підготовки 171 Електроніка

.

Укладач: *Терещенко Тетяна Олександрівна,* докт. техн. наук

Відповідальний редактор: *Ямненко Ю. С.,* завідувач кафедри промислової електроніки, докт техн. наук, проф.

Рецензенти: *Михайлов С.Р.,* доцент кафедри електронних приладів та пристроїв, канд. техн. наук, доц.

Навчальний посібник «Розподілені мікропроцесорні системи. Практичні заняття для підготовки докторів філософії» сприяє набуттю практичних навичок створення розробки апаратної частини та програмного забезпечення розподілених мікропроцесорних систем та методів їх налагодження.

Київ – 2018

ЗМІСТ

ВСТУП

Курс "Розподілені мікропроцесорні систем” є компонентом циклу загальної підготовки фахівців рівня «Доктор філософії» в галузі знань 17 «Електроніка та телекомунікація» за спеціальністю 171 «Електроніка» за спеціалізацією «Електронні системи» за денною формою навчання і відноситься до навчальних дисциплін для здобуття глибинних знань зі спеціальності.

Основними завданнями циклу практичних занять є закріплення теоретичних положень навчальної дисципліни і набуття умінь та досвіду їх практичного застосування шляхом виконання завдань, заданих викладачем.

Практичне заняття 1. Програмні середовища розробок мікропроцессорних систем на базі ARM

Практичне заняття 1.1.

Мета: Здобуття навичок роботи з програмним середовищем розробки мікроконтролерних систем *CooCox*.

Завдання: Кожній бригаді створити власний проект і показати викладачу файл *main.c* .

Теоретичні відомості

CooCox CoIDE - високоінтегроване програмне середовище, призначене для розробки коду мікроконтролерів архітектури ARM. Програма підтримує мікроконтролери серії ST, а також ряд інших сімейств: Atmel, Holtek, Freescale, Nuvoton, NXP, Energy Micro, Texas Instruments і деякі інші. Список чіпів постійно збільшується з кожною версією програми. Убудований дебаггер ST-Link підтримує всі основні режими налагодження.

Останню версію програми можна скачати з офіційного сайту www.coocox.org. Для скачування необхідно зареєструватися, реєстрація проста і безкоштовна. Потім інсталюємо скачаний файл і запускаємо.

Після встановлення та налаштування *CooCox* можна приступати до виконання лабораторних робіт. Здійснивши запуск програми, отримуємо головне меню наступного виду (меню англійською мовою) (рис.1)

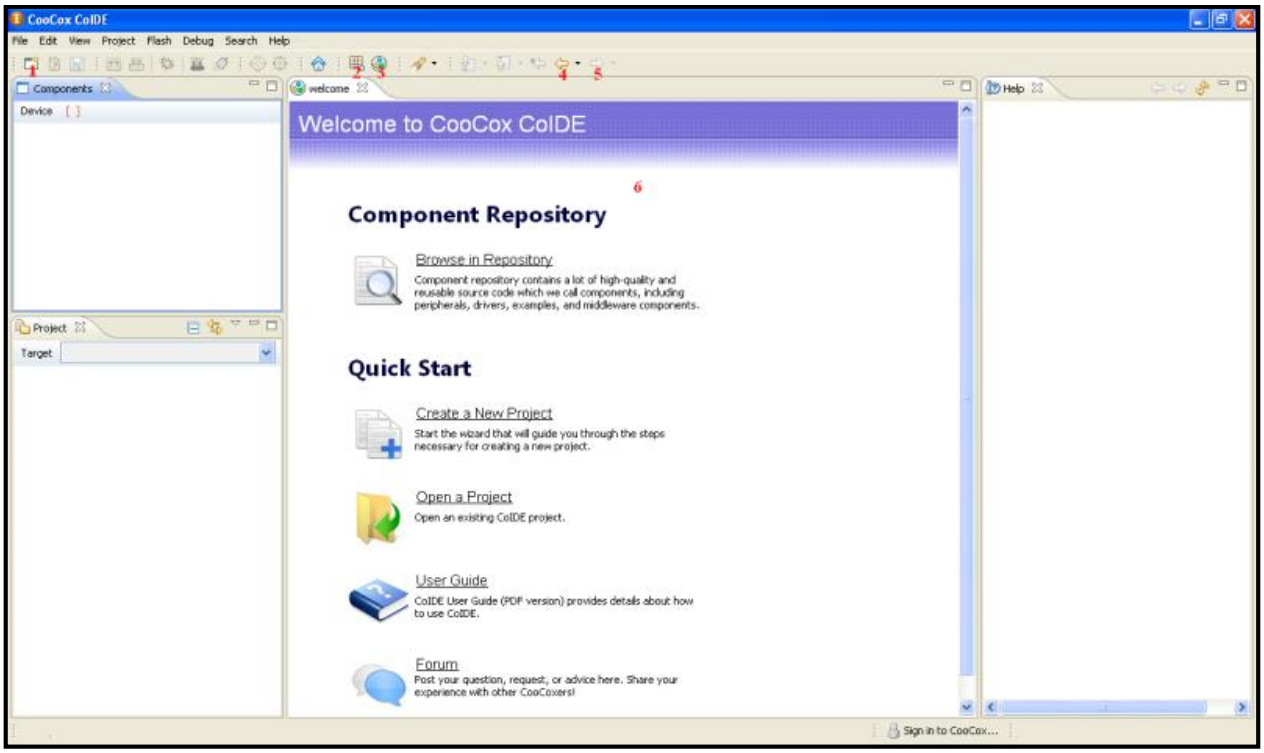

Рис.1. Головне меню *CoIDE*

Червоними цифрами виділені основні піктограми, значення яких відповідно:

1. Створення нового проекту.
2. Відображення меню *Repository*.
3. Відображення меню *Welcome* (рис.1).
4. Повернення на крок назад (аналог переходу між папками у *Windows*).
5. Крок вперед.
6. Головний фрейм для відображення.

Натиснувши на піктограму 1, з'явиться меню (рис. 2), на якому запропоновано обрати ім'я проекту та шлях до нього. Якщо обрати *"Use default path"*, то у рядку *"Project path"* буде вказаний шлях до папки *".../CoIDE/workspace"*.

УВАГА! Назва проекту та шлях до нього має містити лише латинські символи. Приклад: C:\Лаби\Student є недопустимим, адже компілятором не розпізнаються символи кирилиці.

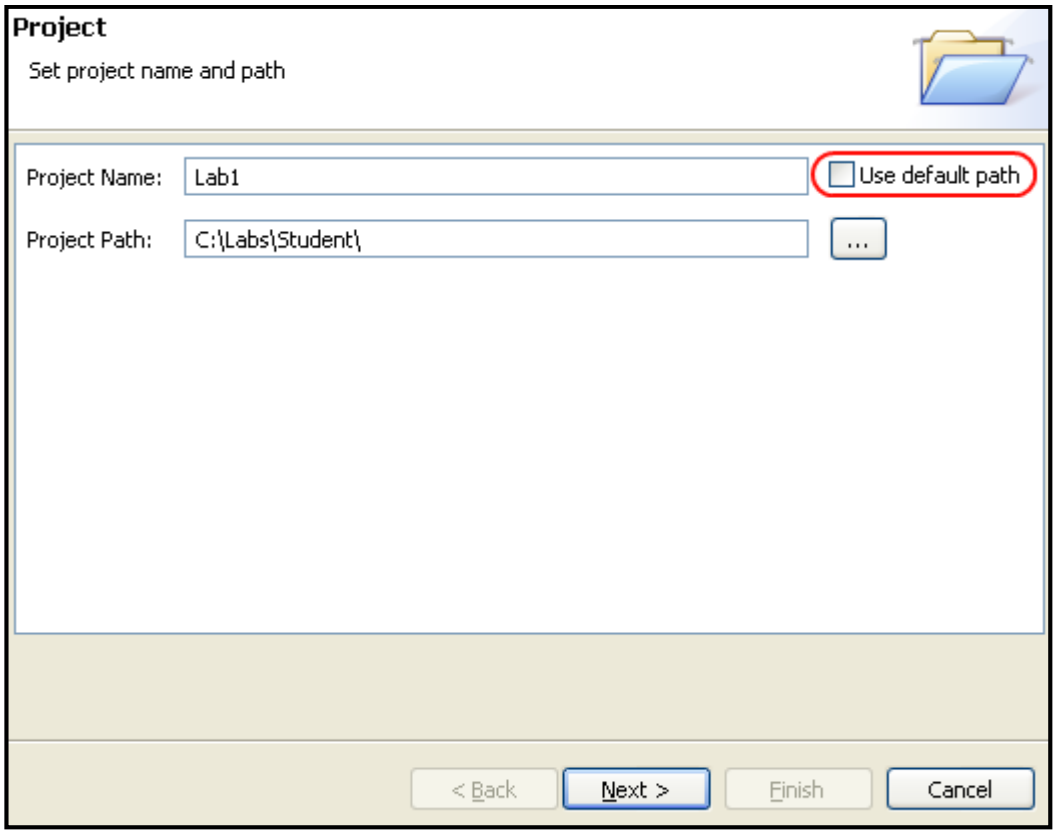

Рис.2. Меню створення проекту

Переходимо до наступного етапу *"Next >"*. На рис.3 пропонується обрати тип пристрою, який підлягає програмуванню.

Chip – обирається для написання, компіляції та завантаження програм, які є універсальними для всіх *Cortex M4* процесорів.

Board – обирається для написання, компіляції та завантаження програм, виконавцем яких є конкретна плата (лабораторний стенд). Для виконання лабораторних робіт обирається цей варіант.

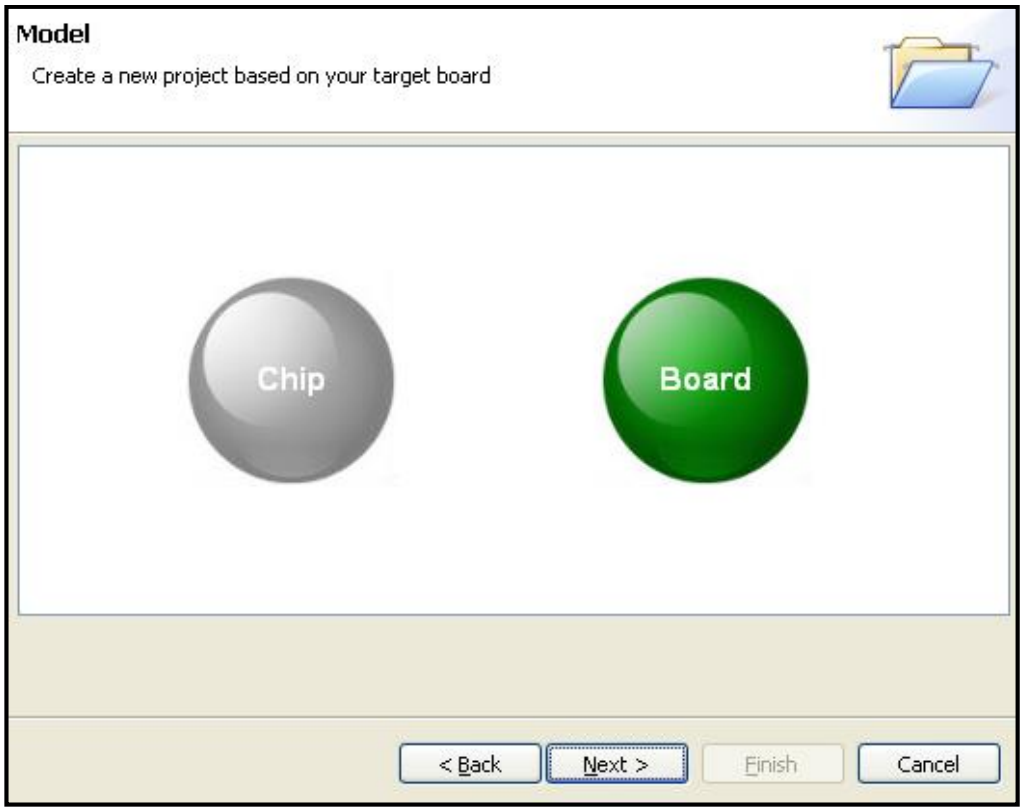

Рис.3. Меню обирання пристрою

Обравши *Board*, на рис. 4. запропоновано обрати необхідну плату. Натискаємо на *CooCox* (тут знаходяться плати, драйвери, для яких написані спільнотою *CooCox.*), обираємо *STM32F4x* (саме такою є серія мікроконтролера, що буде використаний для роботи) та робочу плату *stm32f4-discovery.*

Кнопною "*Finish"* процес створення проекту завершується (рис.5). Наступним кроком є додавання до програми необхідних елементів, а саме: налаштування процесора, його тактової частоти, параметрів завантаження. Для цього необхідно перейти до вкладники *Peripherials* та обрати наступні елементи:

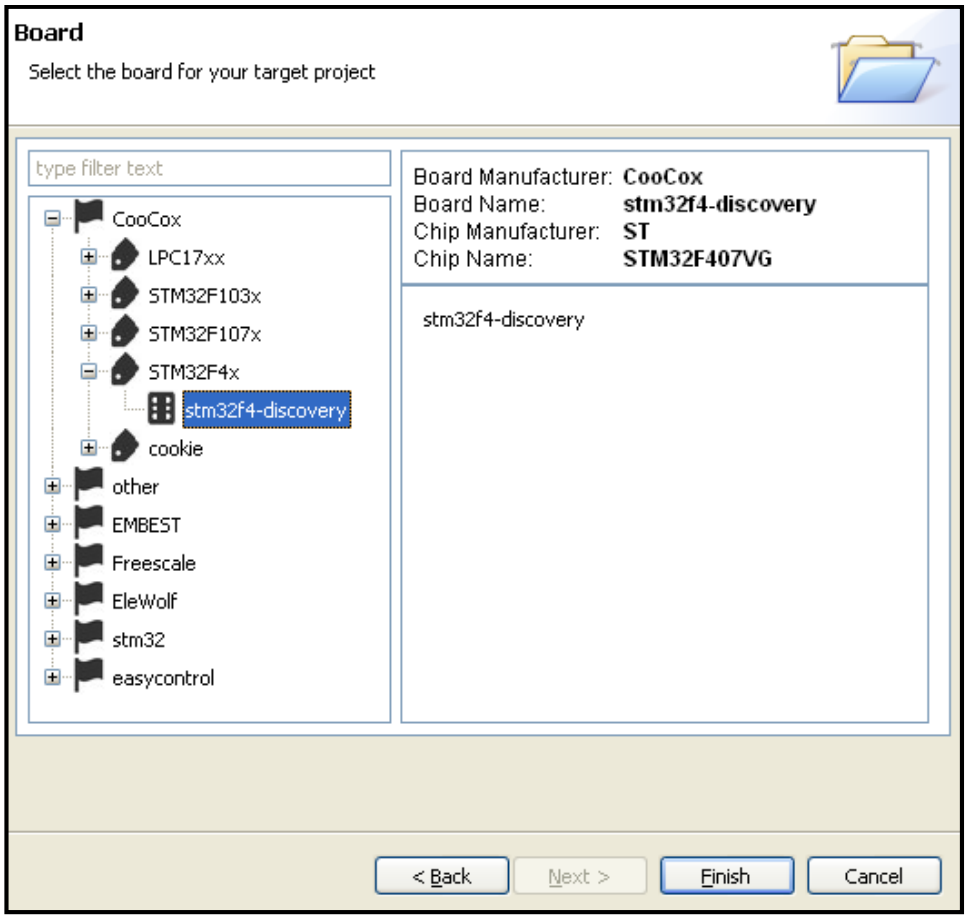

Рис.4. Останній етап створення проекту

- *M4 CMSIS Core. CMSIS (Content Management Interoperability Services)*– це сервіси для роботи з процесором, яку створює компанія для власного компілятора. У даному ПО CMSIS написаний *CooCox* і може потребувати певних змін для іншого ПО (наприклад, *Keil*).
- *CMSIS BOOT*. Даний елемент виконує налаштування запуску МК. Також може бути різним для інших ПО.

Основними елементами середовища розробки є:

1. Побудова проекту (*Build project*). Виконує компіляцію усього коду.
2. Перебудова проекту (*Rebuild project*). Компілює ті частини коду, що були змінені.

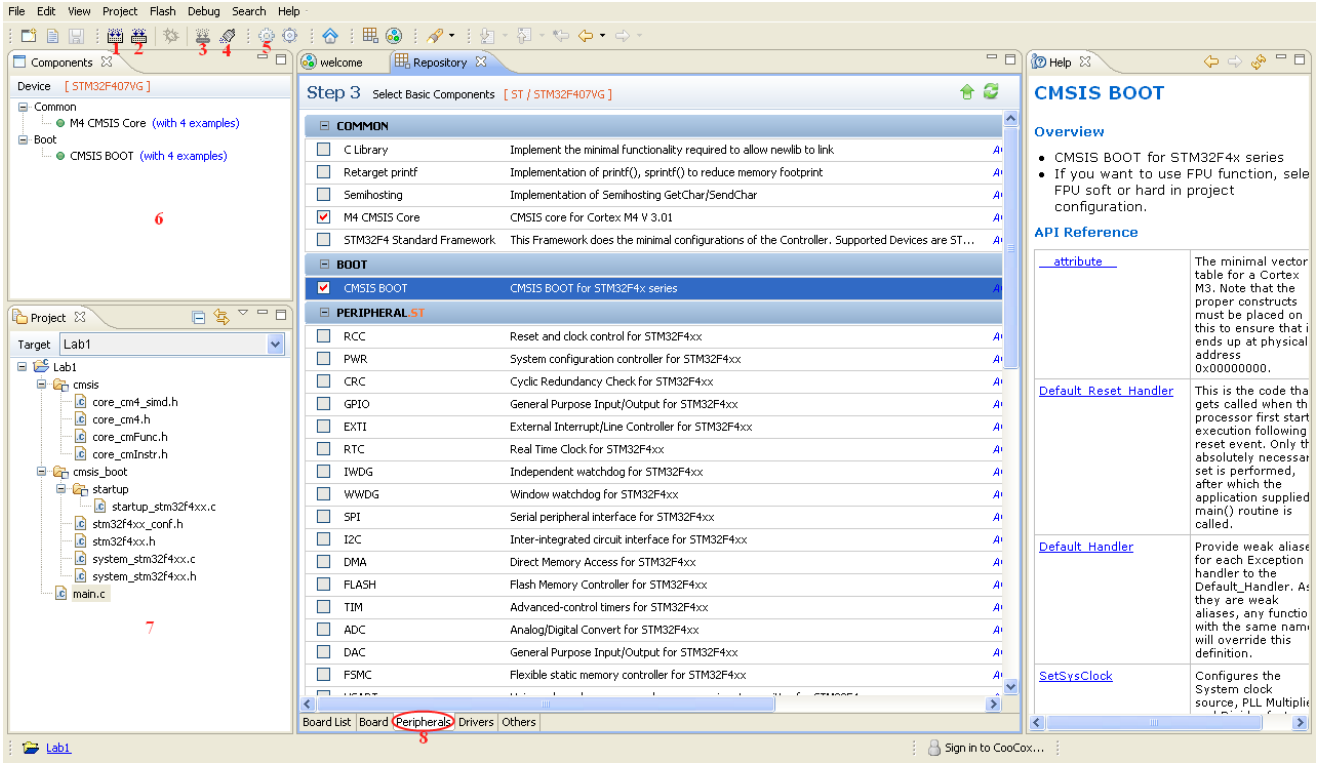

Рис.5. Меню розробки

3. Завантаження коду до МК (*Download Code To Flash*). Виконується програмування МК.

4. Очистка пам’яті МК (*Erase Flash*).

5. Налаштування (*Configuration*).

6. Вікно, що показує підключення бібліотек. Тут відображаються всі об’єкти, що були додані з *Peripherials* та приклади роботи з ними (написані за допомогою макросів *CooCox*).

7. Вікно дерева проекту.

Головний файл, де пишеться основний код програми, має назву *main.c*. Оберіть його з дерева проекту. На цьому дана робота завершена.

Інший шлях створення проекту є наступним: у головному вікні, що зображено на рис.1, натиснути на *Browse in Repository*. Відкриється вікно вибору мікроконтролера. На платі, що є лабораторним стендом, встановлено мікроконтролер *STM32F407VGT6* сімейства *STM32F4xx*. Необхідно обрати необхідний мікроконтролер із заданого списку, після чого буде доступним вікно, що зображено на рис.5.

Одним з недоліків CooCox CoIDE варто відзначити відсутність компілятора GCC, який потрібно завантажити і встановити окремо. Для серії ARM існує кілька варіантів компіляторів з різними наборами допоміжних засобів. За замовчуванням CooCox CoIDE розроблялася для взаємодії з ARM GCC. Останню версію компілятора можна знайти за посиланням https://launchpad.net/gcc-arm-embedded. Праворуч вибираємо тип вашої ОС і скачуємо останню версію. Потім запускаємо файл та інсталюємо gcc_toolchain. Після цього в налаштуваннях CoIDE необхідно вказати правильний шлях до нього. Для цього необхідно відкрити вкладку *Projec*t та вибрати пункт *Select Toolchain Path.*

Увага! Вказувати потрібно директорію *bin*, як зображено на рис. 6.

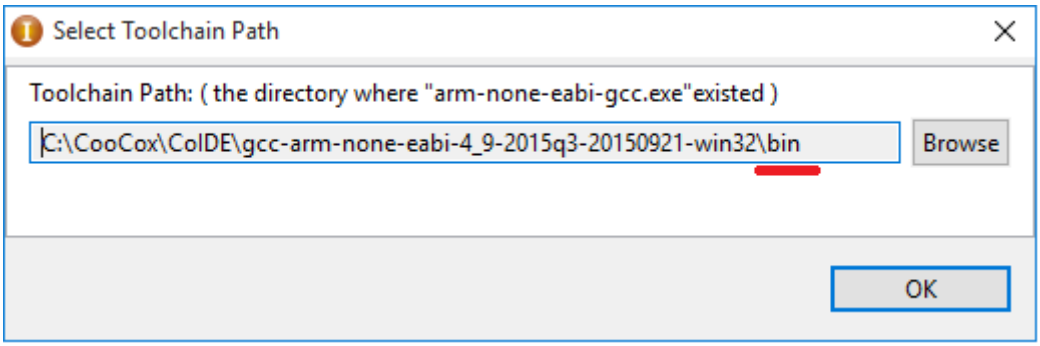

Рис. 6. Вікно для вказування шляху до компілятора

Практичне заняття 1.2.

Мета: Здобуття навичок роботи з програмним середовищем розробки мікроконтролерних систем STM32CubeMX

Завдання: Кожній бригаді створити власний проект і показати викладачу файл *main.c* .

Розглянемо етапи проектування та реалізації прошивок для мікроконтролерів http://electroprog.ru/stm32f4_cubemx_start/

- Установка компілятора;
- Установка бібліотек;
- Вибор мови розробки (ASM, C, C ++);
- Вибор середовища програми: або це бібліотеки, або ОС.

Бібліотеки:

CMSIS, SPL, HAL;

ОС:

CMSIS RTOS, µC / OS-II, FreeRTOS, Linux, Android і інші.

Основні компілятори ARM наступні

- RealView Development Suite (ARM C/C++ Compiler)
- Keil MDK-ARM (ARM C/C++ Compiler)
- IAR Embedded Workbench for ARM (IAR C/C++ Compiler)
- MULTI IDE for ARM (Green Hills C/C++ Compiler)
- CooCox CoIDE (GCC Compiler)

  Далі будемо використовувати Keil https://www.keil.com/download/product/.

STM32CubeMX - це генератор ініціалізації коду для сімейства STM32, що дозволяє автоматично налаштувати всю периферію, для даного кристала або згенерувати код ініціалізації для якогось відлагоджувального набору.
STM32Cube є комплексним програмним рішенням, що комбінує вбудоване програмне забезпечення елементів MCU на базі програмного забезпечення STM32CubeMX. Вбудоване програмне забезпечення не тільки охоплює всі мікроконтролери STM32 з високою переносимістю, драйверів низького рівня, але поставляється з набором компоненти middleware рівня, такі як RTOS, USB, TCP / IP, файлова система або графіка. STM32CubeMX допомагає користувачеві налаштувати STM32 MCU (терморегулятори, ланцюги тактирования і периферію) і програмне забезпечення стеки. Вона також може

допомогти оцінити енергоспоживання завдяки калькулятору розрахунку споживаної потужності. У STM32Cube вбудовані бібліотеки програмного забезпечення і STM32CubeMX генератор коду / конфігуратор може бути використаний незалежно один від одного, але їх повний потенціал досягається коли вони використовуються разом; як тільки MCU налаштований, Користувач може генерувати ініціалізації С код, заснований на вироблених налаштуваннях в STM32CubeMX

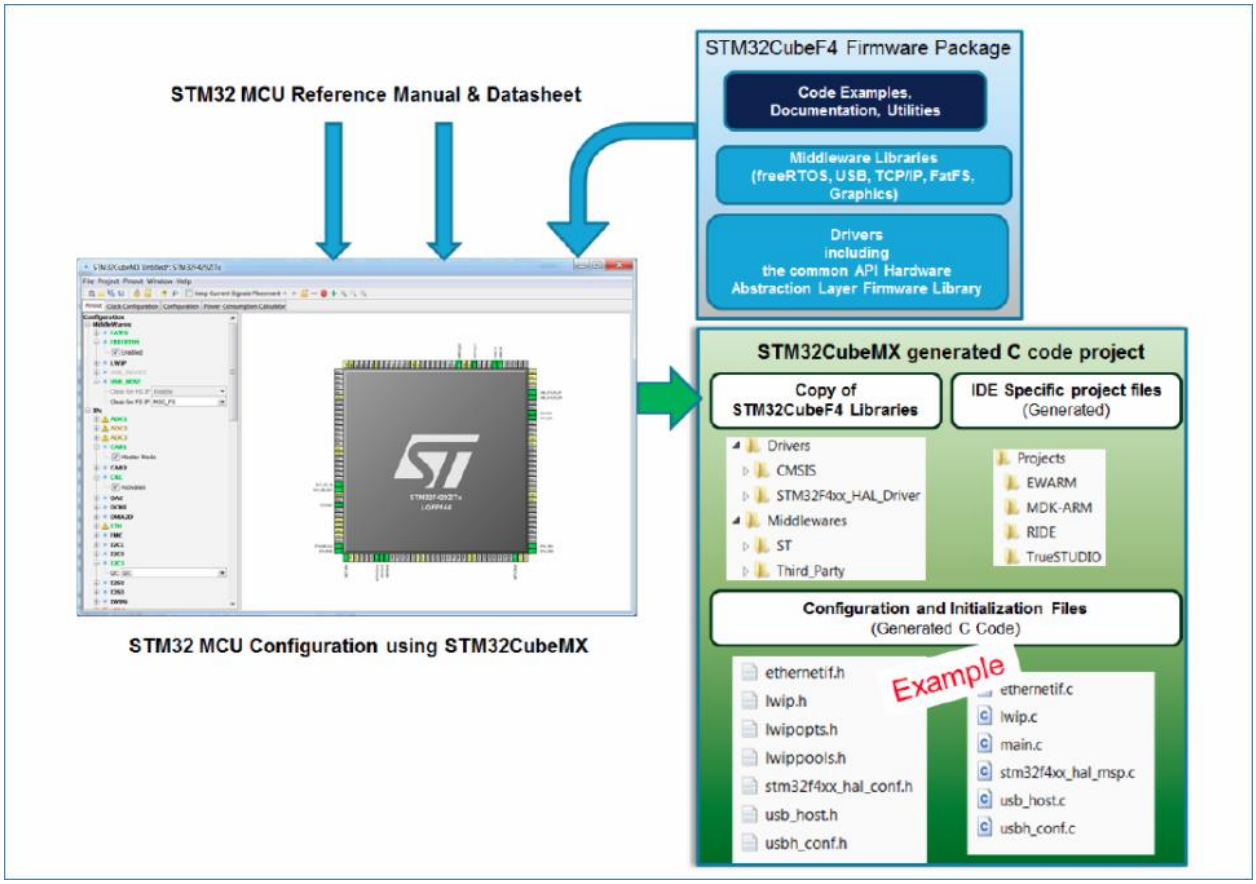

Рис. 7 Налаштування STM32CubeMX

Перед застосуванням STM32CubeMX необхідно проглянути відео по встановленню та створенню перших проектів, наприклад такі:

- Ознайомлення https://www.youtube.com/watch?v=HwomcPSQsRE&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk
- Програма STM32CubeMX

https://www.youtube.com/watch?v=SO83bJ-44OY&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk&index=4

- I2C, дисплей

  https://www.youtube.com/watch?v=XEBVHUNOdx8&index=6&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk

  https://www.youtube.com/watch?v=cVfSoURD1L4&index=7&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk

  https://www.youtube.com/watch?v=7obHbAQ22XU&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk&index=22

  https://www.youtube.com/watch?v=DYaXDQCJDEQ

- АЦП

  https://www.youtube.com/watch?v=zVPb2ZRajxw&index=16&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk

  https://www.youtube.com/watch?v=pqNjWe9HXz8&index=17&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk

  https://www.youtube.com/watch?v=0fpdNWFnggQ&index=18&list=PLJTlt64jBcCuRgAMGImQzxRlYdSKa5Ifk

Завдання на СРС. Переваги та недоліки створення проектів в середовищі CooCox та STM32Cube.

Практичне заняття 2. Особливості використання портів загального призначення GPIO

Мета: Аналіз різних способів написання програми (за допомогою встановлення конкретних бітів; .за допомогою написання HEX значення регістра; за допомогою макросів). Визначення переваг та недоліків кожного способу

Завдання Запрограмувати вказані у завданні виводи портів у вказаний режим роботи з подальшою демонстрацією на платі. Програмний код писати за допомогою регістрів (без використання макросів).

Таблиця 1

Варіант	Порт	Піни на введення	Піни на виведення	Стан	Частота, МГц
1	A	0	1	Push-pull	2
2	B	2	3	Push-pull	25
3	C	4	5	OD, PuUp	50
4	D	6	7	OD, PuUp	100
5	E	8	9	Push-pull	2
6	A	10	11	Push-pull	25
7	B	12	13	OD, PuUp	50
8	C	14	15	OD, PuUp	100
9	D	0	1	Push-pull	2
10	E	2	3	Push-pull	25
11	A	4	5	OD, PuUp	50
12	B	6	7	OD, PuUp	100
13	C	8	9	Push-pull	2
14	D	10	11	Push-pull	25
15	E	12	13	OD, PuUp	50

Розшифровка позначень: *PuUp – pull-up; OD – open-drain.*

Дія, яка має бути виконана, задається викладачем особисто кожній бригаді.

Теоретичні відомості

Існує декілька способів написання програми, а саме:

1. За допомогою встановлення конкретних бітів.
2. За допомогою написання HEX значення регістра.
3. За допомогою макросів.

У даній і наступних лабораторних роботах допускається написання програм лише першими двома методами, адже лише вони дають уявлення про роботу з МК. У прикладі застосовано обидва способи.

Алгоритм налаштування портів наступний:

1. Увімкнення тактування потрібного виводу (RCC−>AHB1ENR регістр).
2. Встановлення направленості виводу (GPIOx−>MODER регістр).
3. Встановлення типу вихідного стану вивода (GPIOx−>OTYPER регістр)
4. Встановлення типу підтяжки (GPIOx−>PUPDR регістр).
5. Встановлення швидкості виводів (GPIOx−>OSPEED регістр).

Для простого виводу (наприклад, запалення діоду) достатньо лише виконати перший пункт.

Лістинг програми для виконання варіанту №15

```
#include "stm32f4xx.h"
int main(void)
{
      SystemInit(); //Конфігурація частот процесова

//Варіант №1 з використанням назв регістрів
      if (RCC_AHB1ENR_GPIOEEN != 1) //Якщо тактування відсутнє
         RCC->AHB1ENR |= RCC_AHB1ENR_GPIOEEN; //Встановити тактування

      GPIOE->MODER   &=~ (GPIO_MODER_MODER12 | GPIO_MODER_MODER13); //Скидання
      бітів

      GPIOE->MODER   |=   GPIO_MODER_MODER13_0; //Встановленян на вихід
      GPIOE->OTYPER  |=   GPIO_OTYPER_OT_13; //Відкритий колектор
      GPIOE->PUPDR   &=~ GPIO_PUPDR_PUPDR13;
      GPIOE->PUPDR   |=   GPIO_PUPDR_PUPDR13_0; //З підтяжкою на джерело
      GPIOE->OSPEEDR &=~ GPIO_OSPEEDER_OSPEEDR13;
      //Встановлення High speed
      GPIOE->OSPEEDR |=   GPIO_OSPEEDER_OSPEEDR13_1;

      //Варіант №2 є HEX варіант
      //RCC->AHB1ENR = 0x00000010;
      //GPIOE->MODER   =  0x04000000;
      //GPIOE->OTYPER  =  0x00002000;
      //GPIOE->PUPDR   =  0x04000000;
      //GPIOE->OSPEEDR =  0x08000000;
    while(1)
    {GPIOE->BSRRL |= GPIO_BSRR_BS_13; //Одиниця на виході
      //GPIOE->BSRRL = 0x2000;
    } }
```

Завдання на СРС. Визначити переваги та недоліки кожного способу написання програми роботи з портами загального призначення GPIO

Практичне заняття 3. Контролер переривань EXTI та NVIC. Таймери.

Мета роботи: Здобуття навичок роботи з налаштуванням тактової частоти процесора, таймерами, перериваннями мікроконтролера, порядком їх налаштування та обробки.

Завдання

Індивідуальні завдання наведені у таблиці 1.

Таблиця 1.

№	Таймер/ канал	Режим	Частота SYSCLK	Частота f_out	γ	Кнопка	Дія
1	2/1	Лічба вгору	10 МГц	1 Гц	0,2	PA1	викладач
2	2/2	Лічба вниз	15 МГц	10 Гц	0,3	PA2	викладач
3	2/3	Лічба вгору	20 МГц	50 Гц	0,4	PA4	викладач
4	2/4	Лічба вниз	25 МГц	100 Гц	0,5	PA5	викладач
5	3/1	Лічба вгору	30 МГц	150 Гц	0,6	PB0	викладач
6	3/2	Лічба вниз	35 МГц	400 Гц	0,7	PB1	викладач
7	3/3	Лічба вгору	40 МГц	1000 Гц	0,8	PB10	викладач
8	3/4	Лічба вниз	45 МГц	1500 Гц	0,7	PB11	викладач
9	4/1	Лічба вгору	50 МГц	5 кГц	0,6	PB14	викладач
10	4/2	Лічба вниз	55 МГц	10 кГц	0,5	PB15	викладач
11	4/3	Лічба вгору	60 МГц	15 кГц	0,4	PC6	викладач
12	4/4	Лічба вниз	65 МГц	50 кГц	0,3	PC7	викладач
13	8/1	Лічба вниз	70 МГц	100 кГц	0,2	PC9	викладач
14	8/2	Лічба вгору	75 МГц	500 кГц	0,3	PD1	викладач
15	8/3	Лічба вниз	80 МГц	1 МГц	0,4	PD2	викладач

Дію, яку необхідно здійснити при натисканні кнопки задається викладачем кожному студентові індивідуально.

Теоретичні відомості

Тактова частота процесора SYSCLK визначається (за замовчуванням) частотою main PLL генератора. За допомогою дільників PLL_M, PLL_N та PLL_P встановлюється необхідне значення SYSCLK. Даний мікроконтролер може працювати на частоті до 180 МГц. Частота, яка використовується main PLL для встановлення потрібного значення, надходить або від зовнішнього кварцового резонатора (HSE 8 … 25 МГц) або від внутрішнього RC генератора (HIS 16 МГц). За наявністю підключеного кварцового резонатора, за замовчуванням, на main PLL надходить HSE. Якщо потрібно змінити джерело main PLL, то для цього необхідно налаштувати RCC(Reset Clock Control) регістри. Функціями, де здійснюється це налаштування, є *SystemInit()* і *SetSystemClock()*, що знаходяться у файлі *system_stm32f4xx.c*.

Формула для розрахунку f_{SYSCLK}:

$$f_{SYSCLK} = \frac{f_{VCO}}{PLL\_P},$$

де f_{VCO} – проміжна частота, яка може бути використана для тактування спеціальної периферії, а $PLL\_P -$ є дільником, значення якого може бути 2,4,6 або 8.

Розрахунок f_{VCO}:

$$f_{VCO} = \frac{HSE \text{ (або } HSI)}{PLL\_M} \cdot PLL\_N \,.$$

Варто взяти до уваги, що:

$$192 \text{ МГц} \leq f_{VCO} \leq 432 \text{ МГц}; \;\; 2 \leq PLL\_M \leq 63; \;\; 192 \leq PLL\_N \leq 432.$$

f_{SYSCLK} є основою AHB1,AHB2,AHB3, APB1 та APB2 генераторів тактової частоти для периферії.

Тактування таймерів TIM2-TIM7, TIM12-TIM14 здійснюється генератором APB1; таймерів TIM1, TIM8-11 – APB2.

Таймери мікроконтролера STM32F4xx мають широкі можливості. У даній лабораторній роботі необхідно буде сформувати ШІМ сигнал на виході каналу таймера потрібної частоти та відповідної гамма. Алгоритм налаштування таймера-лічильника:

1. Увімкнення тактування таймера (RCC−>APB1ENR регістр).
2. Налаштування режиму лічби (TIMx−>CR1 регістр).
3. Налаштування дільника (TIMx−>PSC регістр).

Увага! Варто врахувати, що значення PSC автоматично буде зменшено на одиницю.

4. Налаштування регістру переповнення (TIMx−>ARR регістр).

Увага! Варто врахувати, що значення ARR автоматично буде збільшено на одиницю.

5. Встановлення переривання таймеру по переповненню (TIMx−>DIER регістр).
6. Запуск таймера (TIMx−>CR1 регістр).

Увага! Функція обробки переривань має містити текст скидання біту, адже автоматичного скидання не відбувається й повторної обробки не відбудеться.

```
void TIMx_IRQHandler(void){
      if((TIMx->SR & TIM_SR_UIF) != 0){
            ...
            TIMx->SR &=~ TIM_SR_UIF;
      }
      }
```

Для обробки переривання необхідно, щоб NVIC (Nested Vectored Interrupt Controller) контролер сприйняв дане програмне переривання та сформував дозвіл на його обробку. Для кожного таймера є своя лінія IRQ, яка має власний номер та пріоритет, встановлений у таблиці векторів переривань IRQ (див. додаток А). Дозвіл на обробку переривань здійснюється за допомогою функції CMSIS *NVIC_EnableIRQ(_IRQn)*, де *_IRQn* – назва лінії переривання.

Налаштування зовнішнього переривання здійснюється за допомогою EXTI (External interrupt/event controller) контролера. EXTI призначений для генерації запиту переривань (IRQ або EVENT) внаслідок прийому зовнішніх сигналів. Налаштування цього контролеру у випадку генерації IRQ передбачає наступні кроки:

- обрання лінії переривання. Кожній групі виводів властивий певний канал EXTI (див. додаток В);
- налаштувати даний вивід як приймача сигналу для генерації переривання (SYSCFG−>SYSCFG_EXTICRX, де X – номер лінії), перед цим ввімкнувши тактування SYSCFG (RCC−>APB2ENR регістр);
- встановлення дозволу на генерування переривання та ввімкнення контролеру (EXTI−>EXTI_IMR регістр);
- встановлення фронту, за яким виконується генерація переривання (EXTI−>EXTI_RTSR та EXTI−>EXTI_FTSR);

Після налаштування лінії ETXI - має бути пристрій введення інформації. У межах даної лабораторної роботи пристроєм введення інформації є кнопка. Відповідно до цього, пін для кнопки має бути налаштований на вхід з підтяжкою.

Приклад виконання варіанту №15:

У файлі *system_stm32f4xx.c* необхідно змінити:

```
#define PLL_M      8
#define PLL_N      320
#define PLL_P      4
```

Файл *main.c*:

```
#include "stm32f4xx.h"

char PWM_Counter=0;

void PORT_INIT(void);
void TIMER_INIT(void);
void INTERRUPT_INIT(void);

int main(void)
{
SystemInit();
PORT_INIT();
```

```
TIMER_INIT();
INTERRUPT_INIT();
while (1)
{
}
}

void TIM8_UP_TIM13_IRQHandler(void){
      if((TIM8->SR & TIM_SR_UIF) != 0){
      ////
      GPIOC->ODR ^= GPIO_ODR_ODR_8;
      ///
      TIM8->SR &=~ TIM_SR_UIF;
      }
      }

      void PORT_INIT()
      {
      RCC->AHB1ENR |= (RCC_AHB1ENR_GPIODEN | RCC_AHB1ENR_GPIOCEN |
      RCC_AHB1ENR_GPIOAEN);
      GPIOC->MODER |= (GPIO_MODER_MODER8_0);
      GPIOD->MODER |= (GPIO_MODER_MODER12_0);
      }
      void TIMER_INIT()
      {
      RCC->APB2ENR |= RCC_APB2ENR_TIM8EN;
      TIM8->CR1 = TIM_CR1_CMS_0;
      TIM8->CR1 = TIM_CR1_DIR;
      TIM8->CR1 = TIM_CR1_ARPE;
      TIM8->PSC = 4-1;
      TIM8->ARR = 10;
      TIM8->DIER = TIM_DIER_UIE;
      TIM8->CR1 = TIM_CR1_CEN;
      }

      void INTERRUPT_INIT(void)
      {
      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
      SYSCFG->EXTICR[1] &=~ SYSCFG_EXTICR1_EXTI0_PA;

      EXTI->IMR |= EXTI_IMR_MR0;
      EXTI->RTSR |= EXTI_RTSR_TR0;

      NVIC_EnableIRQ(EXTI0_IRQn);
      NVIC_EnableIRQ(TIM8_UP_TIM13_IRQn);
      }

      void EXTI0_IRQHandler(void)
      {
            if((EXTI->PR & EXTI_PR_PR0) != 0)//EXTI_Line_0 0x000001
            {
            GPIOD->ODR ^= GPIO_ODR_ODR_12; //Реверс бітів
            EXTI->PR |= EXTI_PR_PR0;
            }
            }
```

Завдання на СРС. Наведіть формули для розрахунку частот тактування процесора та периферії Охарактеризуйте бібліотечні функції контролерів переривань NVIC та EXTI . Охарактеризуйте бібліотечні функції таймерів загального призначення

Практичне заняття 4. Інтерфейс UART /USART

Мета: Здобуття навичок роботи з інтерфейсом USART у мікроконтролерах STM Використання бібліотечних функцій

Завдання: написання програми керування світінням кожного з чотирьох світлодіодів. По команді з ПК відбувається запалення чи гасіння відповідного світло діода.

Параметри налаштування USART наведені у таблиці 1.

Таблиця 1

№	USART name	Baud rate	Parity	Stop bits
1	USART6	600	none	0.5
2	USART2	1200	odd	1
3	USART3	2400	even	1.5
4	UART4	4800	none	2
5	UART5	9600	odd	0.5
6	USART6	14400	even	1
7	USART5	19200	none	1.5
8	USART2	28800	odd	2
9	USART3	38400	even	0.5
10	UART4	56000	none	1
11	UART5	57600	odd	1.5
12	USART6	115200	even	2
13	USART4	128000	none	1
14	USART2	256000	odd	1.5
15	USART3	9600	even	2

Теоретичні відомості:

Алгоритм налаштування інтерфейсу USART у мікроконтролерах STM32 складається з наступних кроків:

1. Виконати ініціалізацію тактової частоти мікроконтролера.
2. Подати тактування на відповідні виводи мікроконтролера.
3. Налаштувати виводи за альтернативною функцією.
4. Подати тактування на контролер USART.
5. Налаштувати параметри передачі.

6. Ввімкнути передавач та приймач.
7. Дозволити переривання по прийому.
8. Ввімкнути USART.

В прикладі розглянемо передачу та приймання даних з іншого пристрою (ПК), для перевірки роботи МК відсилає прийняті дані назад на ПК.

Для передачі даних з ПК рекомендовано використовувати програму Terminal (рис. 1).

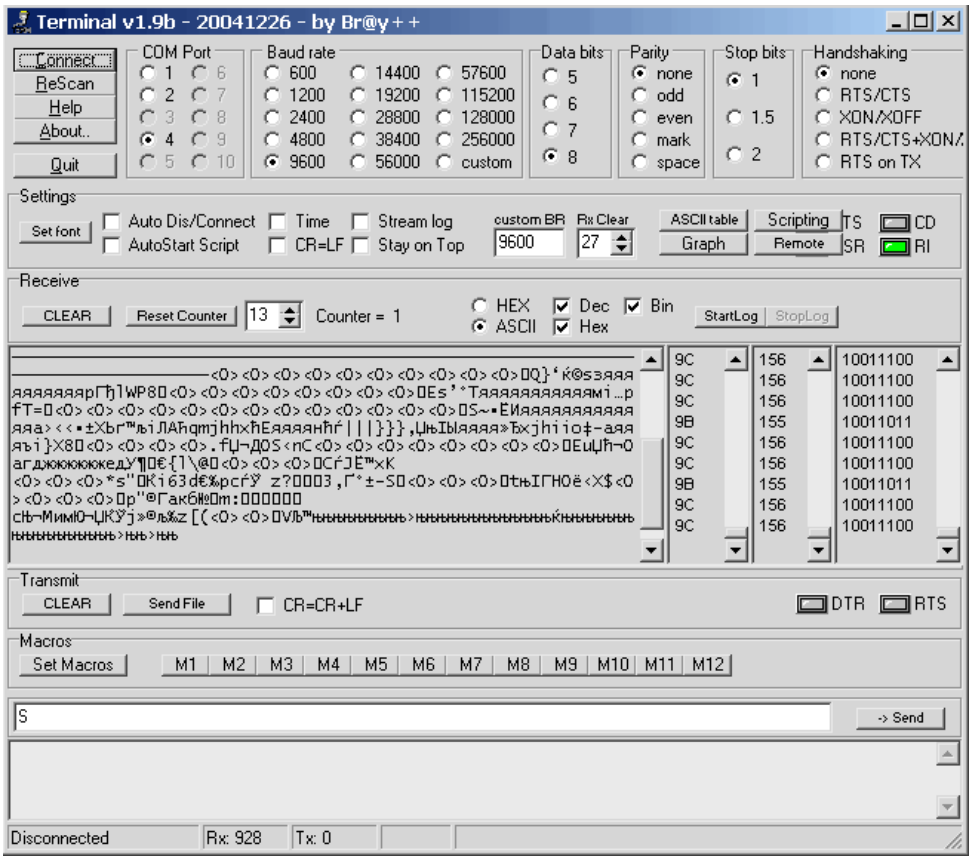

Рис. 1 Вікно програмиTerminal

Лістинг програми:

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_usart.h"
#include "misc.h"

uint8_t data;
int main(void)
{

	//включаємо тактування потрібних пристроїв
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
```

```
        //ініціалізація порта виходів і призначення виконання альтернативної функції
        GPIO_InitTypeDef GPIO_InitStruct;
        GPIO_InitStruct.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOD, &GPIO_InitStruct);

       // вибираємо альтернативну функцію виконуваєму виводами порта
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource5, GPIO_AF_USART2);
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource6, GPIO_AF_USART2);

        //ініціалізація USaRT
        USART_InitTypeDef USART_InitStruct;
        USART_InitStruct.USART_BaudRate = 9600;
        USART_InitStruct.USART_WordLength = USART_WordLength_8b;
        USART_InitStruct.USART_StopBits = USART_StopBits_1;
        USART_InitStruct.USART_Parity = USART_Parity_No;
        USART_InitStruct.USART_HardwareFlowControl =USART_HardwareFlowControl_None;
        USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
        USART_Init(USART2, &USART_InitStruct);

        //налаштування переривання, ввімкнення USART
        USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
        NVIC_InitTypeDef NVIC_InitStructure;
        NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
        USART_Cmd(USART2, ENABLE);

    while(1)
    {
    ;
    }
}

void USART2_IRQHandler(void)
{
       if( USART_GetITStatus(USART2, USART_IT_RXNE) )
  {
              USART_ClearITPendingBit(USART2, USART_IT_RXNE);
              data = USART_ReceiveData(USART2);
              }

       USART_SendData(USART2, data);
}
```

Завдання на СРС. Охарактеризуйте бібліотечні функції роботи з інтерфейсом USART

Практичне заняття 5. Інтерфейс SPI

Мета роботи: Здобуття навичок роботи з інтерфейсом SPI у мікроконтролерах STM Використання бібліотечних функцій для керування процесом обміну даними між двома контроллерами.

Завдання:

Одному з контролерів на вхід дається аналогова напруга. Він запускає АЦП і в залежності від отриманого значення видає на шину SPI значення. Індивідуальні завдання наведені у таблиці 1. Другий контролер в залежності від отриманих даних запалює світлодіоди. Якщо приймає 0x01 - запалює один діод, якщо приймає 0x02 - запалює два і т.д.

Таблиця 1

№	Напруга	Відправлене значення
13	0 < U < 1В 1В < U < 2В 2В < U < 3В 3В < U < 3.3В 3.3В < U	0x00 0x01 0x02 0x03 0x04

Приклад виконання варіанту №13:

Для створення проекту для роботи з SPI необхідно додати файли з SPL. Спочатку напишемо програму для ведучого (SPI Master)

```
/*****************************spi master.c*****************************/
#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_spi.h"
#include "stm32f10x_adc.h"

/*********************************************************************/
//оголошення змінних
GPIO_InitTypeDef port;
SPI_InitTypeDef spi;
ADC_InitTypeDef adc;
uint8_t sendData;
uint16_t counter;
uint16_t data;

/*********************************************************************/
void initAll()
{
	// Ініціалізація
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

```
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

	port.GPIO_Mode = GPIO_Mode_AF_PP;
	port.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
	port.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &port);

	ADC_StructInit(&adc);
	adc.ADC_ContinuousConvMode = ENABLE;
	adc.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
	ADC_Init(ADC1, &adc);

	SPI_StructInit(&spi);
	spi.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	spi.SPI_Mode = SPI_Mode_Master;
	spi.SPI_DataSize = SPI_DataSize_8b;
	spi.SPI_CPOL = SPI_CPOL_Low;
	spi.SPI_CPHA = SPI_CPHA_2Edge;
	spi.SPI_NSS = SPI_NSS_Soft;
	spi.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;
	spi.SPI_FirstBit = SPI_FirstBit_MSB;
	spi.SPI_CRCPolynomial = 7;
	SPI_Init(SPI1, &spi);

	GPIO_StructInit(&port);
	port.GPIO_Mode = GPIO_Mode_IPD;
	port.GPIO_Pin = GPIO_Pin_0;
	port.GPIO_Speed = GPIO_Speed_2MHz;
	GPIO_Init(GPIOA, &port);
}


/*****************************************************************************/
int main()
{
	__enable_irq();
	initAll();
	// Вмикання АЦП
	ADC_Cmd(ADC1, ENABLE);
	ADC_SoftwareStartConvCmd(ADC1, ENABLE);
	// Вмикання SPI
	SPI_Cmd(SPI1, ENABLE);
	while(1)
	{
		// Лічильник з довільним числом
		counter++;
		data = ADC_GetConversionValue(ADC1);
		// Аналіз даних через АЦП
		if (data == 0xFFF)
		{
			sendData = 0x04;
		}
		else if (data > 0xE8B)
		{
			sendData = 0x03;
		}
		else if (data > 0x9B2)
		{
			sendData = 0x02;
		}
		else if (data > 0x4D9)
		{
			sendData = 0x01;
```

```
            }
            else
            {
                sendData = 0x00;
            }
            if(counter == 15000)
            {
                // Відправка даних
                SPI_I2S_SendData(SPI1, sendData);
            }
        }
}

/*****************************End of file*****************************/
```

Програма для SPI Slave.

```
/*****************************spi slave.c*****************************/
#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_spi.h"


/*********************************************************************/
GPIO_InitTypeDef port;
SPI_InitTypeDef spi;
uint8_t data;
uint8_t needUpdate;


/*********************************************************************/
void initAll()
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    port.GPIO_Mode = GPIO_Mode_AF_PP;
    port.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    port.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &port);

    port.GPIO_Mode = GPIO_Mode_Out_PP;
    port.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
    port.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &port);

    SPI_StructInit(&spi);
    spi.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    spi.SPI_Mode = SPI_Mode_Slave;
    spi.SPI_DataSize = SPI_DataSize_8b;
    spi.SPI_CPOL = SPI_CPOL_Low;
    spi.SPI_CPHA = SPI_CPHA_2Edge;
    spi.SPI_NSS = SPI_NSS_Soft;
    spi.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;
    spi.SPI_FirstBit = SPI_FirstBit_MSB;
    spi.SPI_CRCPolynomial = 7;
    SPI_Init(SPI2, &spi);
}



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

```
int main()
{
	__enable_irq();
	initAll();
	SPI_Cmd(SPI2, ENABLE);
	NVIC_EnableIRQ(SPI2_IRQn);
	// Дозвіл на переривання
	SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
	// Вмикання відповідних світлодіодів
	while(1)
	{
		if (needUpdate == 1)
		{
			GPIO_ResetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3);
			if (data == 0x01)
			{
				GPIO_SetBits(GPIOA, GPIO_Pin_0);
			}
			if (data == 0x02)
			{
				GPIO_SetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1);
			}
			if (data == 0x03)
			{
				GPIO_SetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2);
			}
			if (data == 0x04)
			{
				GPIO_SetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3);
			}
			needUpdate = 0;
		}
	}
}


/*****************************************************************/
void SPI2_IRQHandler()
{
	data = SPI_I2S_ReceiveData(SPI2);
	needUpdate = 1;
}


/*****************************End of file*****************************/
```

В даному прикладі вся робота з SPI полягає у використанні переривань. При встановленні прапору, що сигналізує про прийняття байту, зчитаємо прийняті данні в перериванні. Після прийняття даних потрібно обновити стан світлодіодів – встановивши змінну needUpdate = 1.

Завдання на СРС. Охарактеризуйте бібліотечні функції роботи з інтерфейсом SPI

Практичне заняття 6. Інтерфейс I2C

Мета: Здобуття навичок роботи з інтерфейсом I2C у мікроконтролерах STM Використання бібліотечних функцій роботи з інтерфейсами I2C та TWI.

Завдання Переслати код АЦП по інтерфейсу I2C на 16x2 LCD дисплей

Теоретичні відомості.

Порти що підтримують обробку даних за протоколом I2C наведено на рис.1

Рис. 1. Характеристики портів

Для створення проекту необхідно мати програми: STM32CubeMX, Keil µVision та Бібліотку HAL для роботи з дисплеєм.

Порядок виконання роботи:

В STM32CubeMX створюємо проект, рис.2.

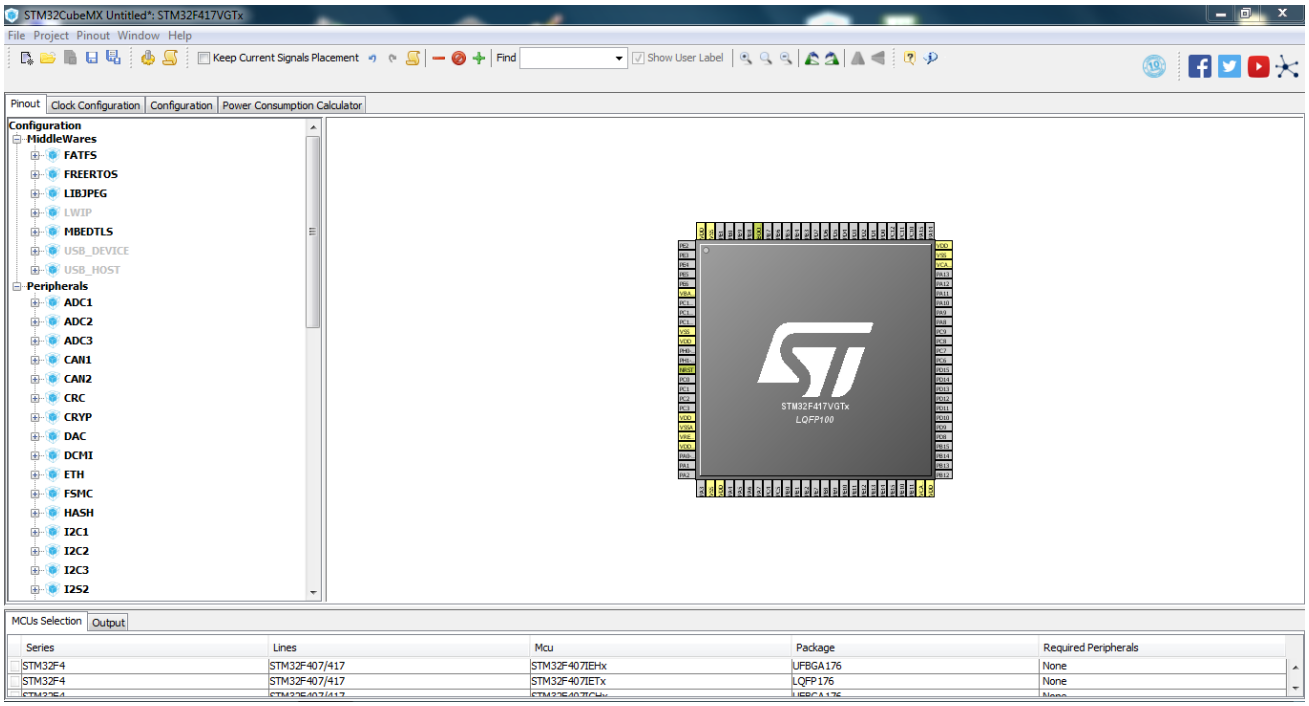

Рис 2 Вікно STM32CubeMX перед початком роботи

Обираємо перший АЦП та 5 вхід, рис.3

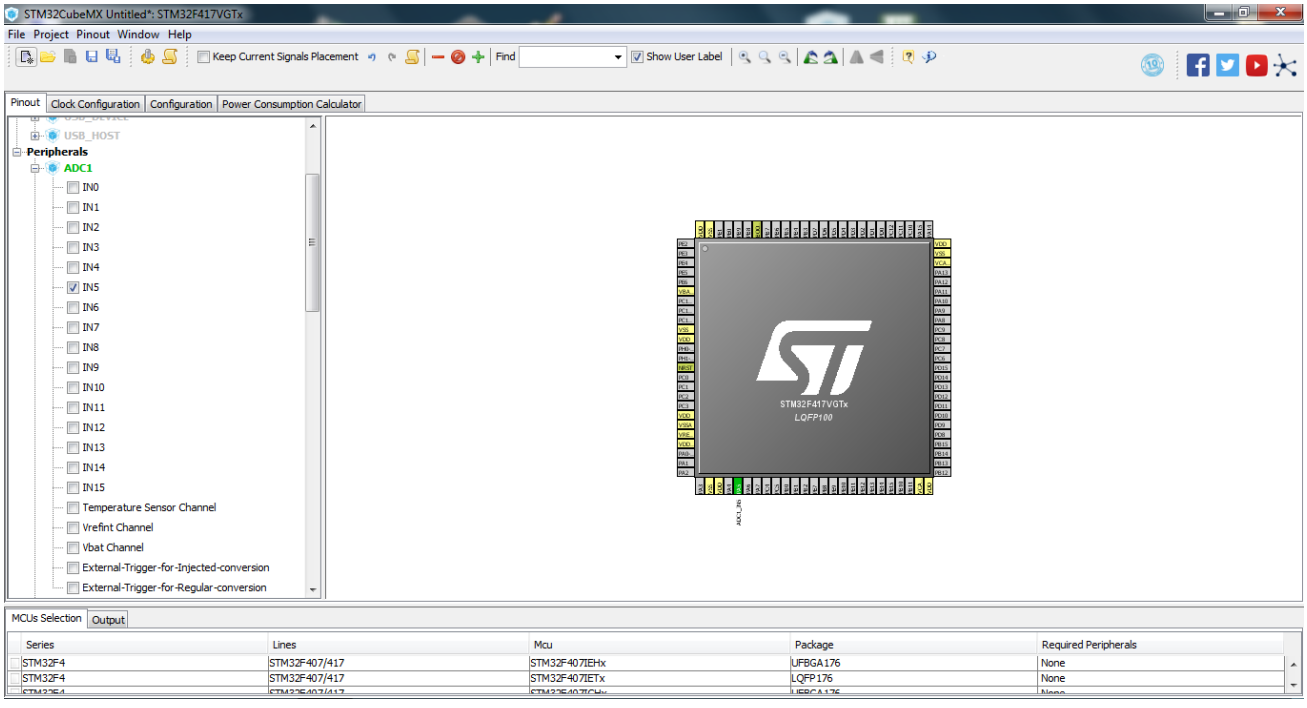

Рис 3 Вікно STM32CubeMX після вибору АЦП

Вигляд вікна програми після вибору входу АЦП наведено на рис.4

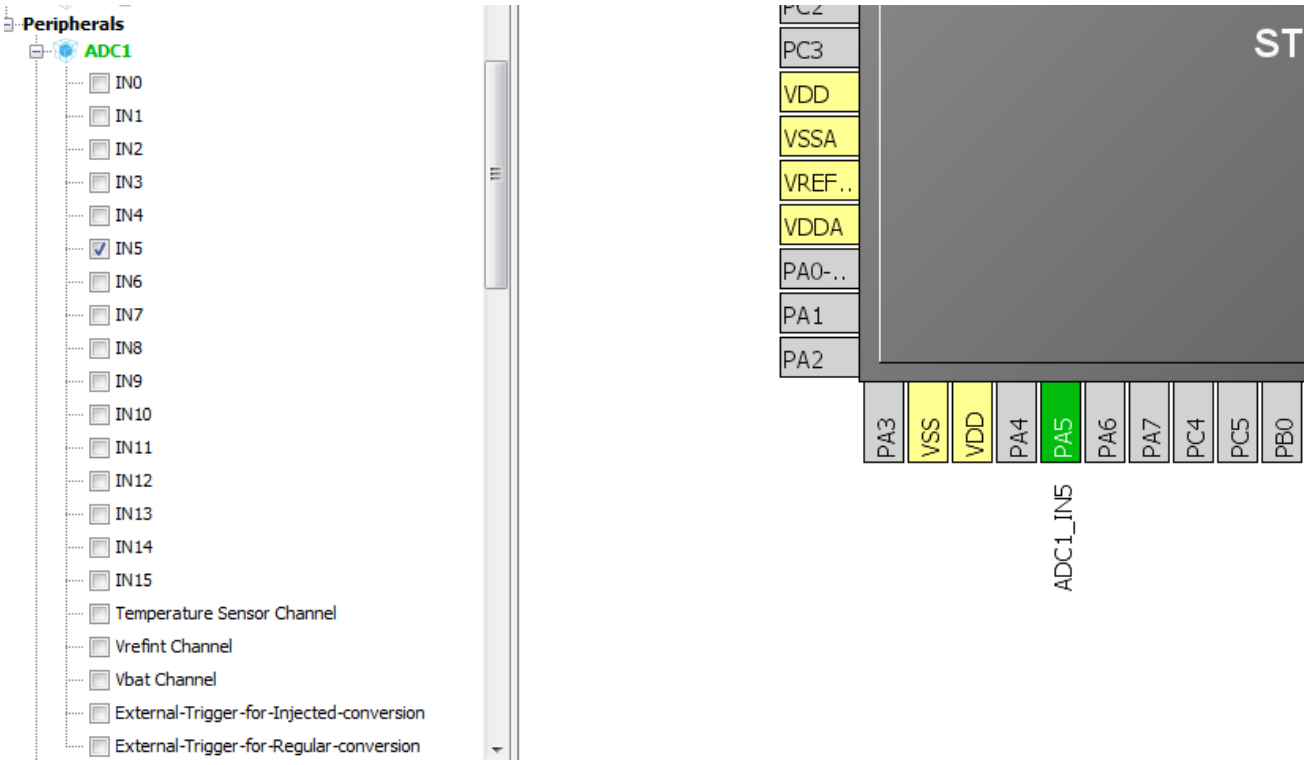

Рис.4

На цей вхід підключимо середній вивід потенцометра, інші 2 вивода - до +3V та GND.

Меню конфігурації матиме вигляд, представлений на рис. 5 та рис.6.

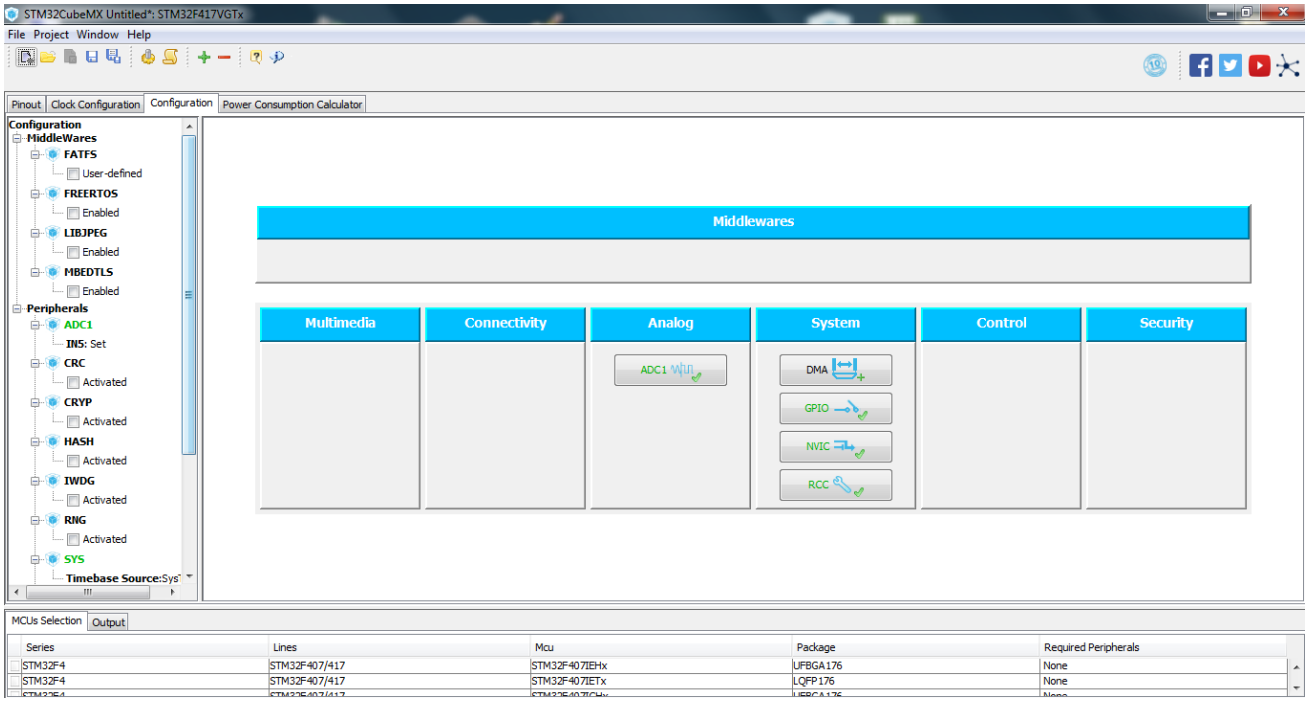

Рис.5

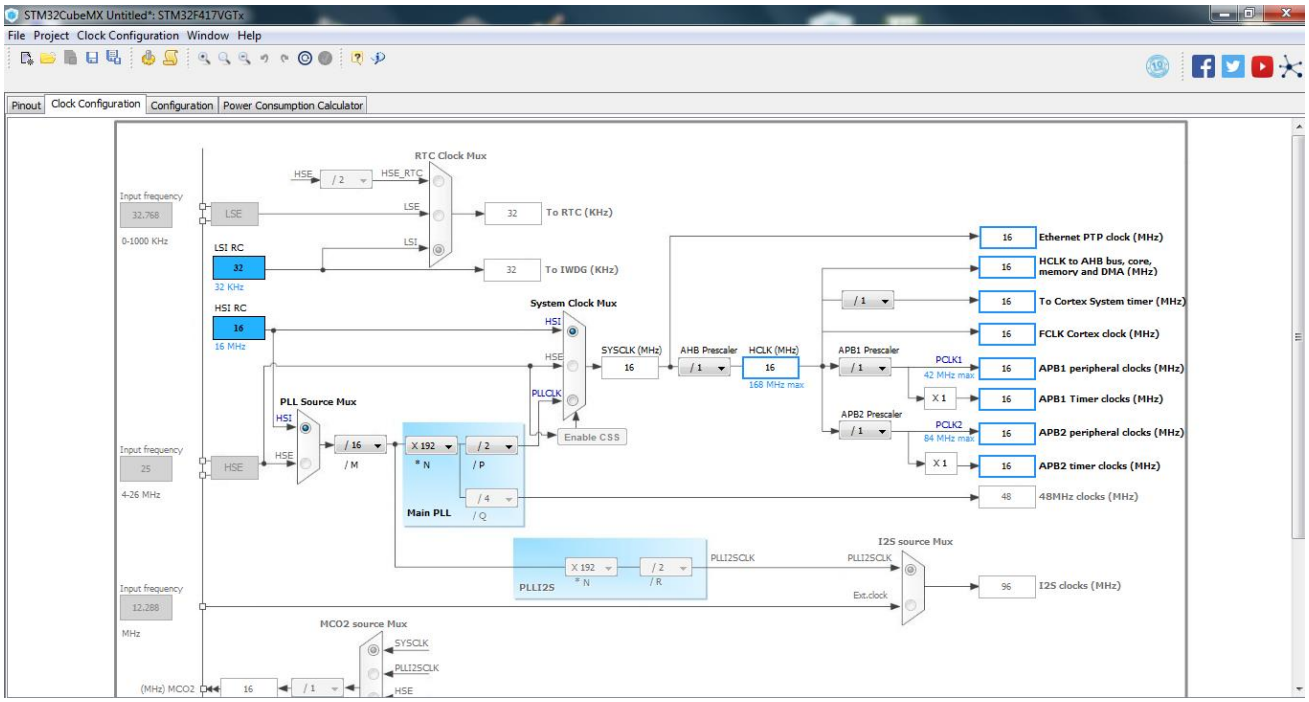

Рис.6

Налаштування АЦП здійснемо, клікнувши по ньому – відкриється меню підлаштувань (рис.7)

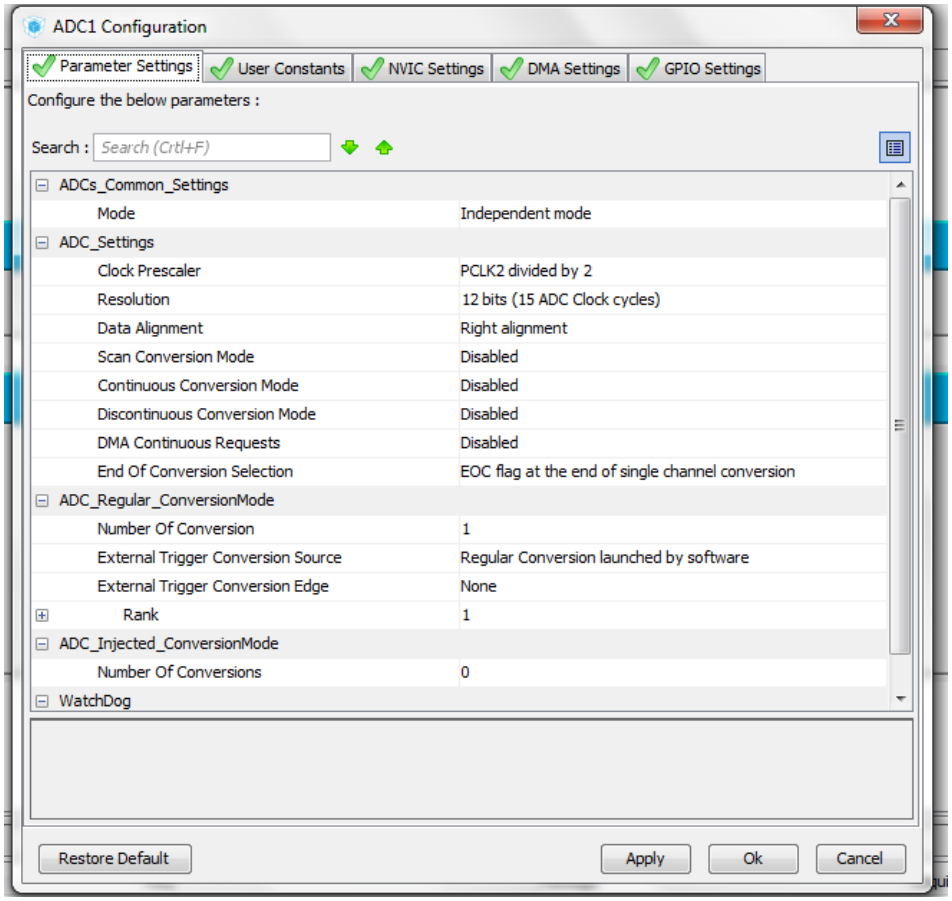

Рис. 7

Далі потрібно в Keil µVision створити необхідні змінні та згенерувати проект

```
34
35 int main(void)
36 {
37
38   /* USER CODE BEGIN 1 */
39   float u;
40   uint16_t i=0;
41   char str[9];
42   /* USER CODE END 1 */
43
44   /* MCU Configuration----------------
45
46   /* Reset of all peripherals, Initiali
47   HAL_Init();
48
```

float u;

uint16_t i=0;

char str[9];

Наприклад так оформити вивдення тексту.

```
66    LCD_ini();
67    LCD_SetPos(5, 0);
68    sprintf(str,"Hello");
69    LCD_String(str);
70    HAL_Delay(500);
71    LCD_SetPos(1, 1);
72    sprintf(str,"Test 12bit ADC");
73    LCD_String(str);
74    HAL_Delay(3000);
75    LCD_Clear ();
76    LCD_SetPos(2, 0);
77    sprintf(str,"Stm32f407VG");
78    LCD_String(str);
79    HAL_Delay(2000);
80    LCD_Clear ();
81    LCD_SetPos(0, 1);
82    sprintf(str,"4095 -MAX Number");
83    LCD_String(str);
84    LCD_SetPos(0, 0);
85    sprintf(str,"0000 -Now Number");
86    LCD_String(str);
87    HAL_Delay(3000);
88    LCD_Clear ();
89    LCD_SetPos(5, 0);
90    sprintf(str,"-Now Number");
91    LCD_String(str);
92    LCD_SetPos(5, 1);
93    sprintf(str,"-Real Volt");
94    LCD_String(str);
```

LCD_ini();

LCD_SetPos(5, 0);

sprintf(str,"Hello");

LCD_String(str);

HAL_Delay(500);

```
LCD_SetPos(1, 1);
sprintf(str,"Test 12bit ADC");
LCD_String(str);
HAL_Delay(3000);
LCD_Clear ();
LCD_SetPos(2, 0);
sprintf(str,"Stm32f407VG");
LCD_String(str);
HAL_Delay(2000);
LCD_Clear ();
LCD_SetPos(0, 1);
sprintf(str,"4095 -MAX Number");
LCD_String(str);
LCD_SetPos(0, 0);
sprintf(str,"0000 -Now Number");
LCD_String(str);
HAL_Delay(3000);
LCD_Clear ();
LCD_SetPos(5, 0);
sprintf(str,"-Now Number");
LCD_String(str);
LCD_SetPos(5, 1);
sprintf(str,"-Real Volt");
LCD_String(str);
```

Також описуємо саму функцію АЦП та виводу зображення на дисплей.

```
/* USER CODE BEGIN WHILE */
while (1)
{
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1,100);
i=HAL_ADC_GetValue(&hadc1);
u=((float)HAL_ADC_GetValue(&hadc1)*3/4096);
HAL_ADC_Stop(&hadc1);
sprintf(str,"%04d",i);
LCD_SetPos (0,0);
LCD_String(str);
HAL_Delay(500);
sprintf(str,"%2.fv",u);
LCD_SetPos (0,1);
LCD_String(str);
}
/* USER CODE END 3 */
```

HAL_ADC_Start(&hadc1);

HAL_ADC_PollForConversion(&hadc1,100);

i=HAL_ADC_GetValue(&hadc1);

u=((float)HAL_ADC_GetValue(&hadc1)*3/4096);

HAL_ADC_Stop(&hadc1);

sprintf(str,"%04d",i);

LCD_SetPos (0,0);

LCD_String(str);

HAL_Delay(500);

sprintf(str,"%2.fv",u);

LCD_SetPos (0,1);

LCD_String(str);

Далі скомпілювавши проект та записав його в ядро, можна побачити, що при зміні опору потенціометра будуть змінюватися рівні напруги. Потенціометр варто взяти не менше 50К Ом.

Завдання на СРС. Охарактеризуйте бібліотечні функції роботи з інтерфейсами I2C та TWI.

Практичне заняття 7. Інтерфейс USB та USB OTG

Мета: Здобуття навичок роботи з інтерфейсами USB та USB OTG Використання бібліотечних функцій**.**

Завдання Використання USB OTG в режимі MCS_HOST. Підключити USB флеш і з нею працювати через OTG порт плати Discovery.

Теоретичні відомості

Бібліотека FatFs (USB версія) має тільки дві функції, які будемо використовувати у цій роботі - http://cxem.net/mc/mc366.php:

1. Функція ініціалізації, яка повинна викликатися тільки на початку.

2. HOST_Do-функція, яка запускається циклічно і повертає статус USB порту. Статус змінюється, як тільки буде виявлений ключ USB флешки, доступною для роботи з FatFs.

Для підключення до плати необхідний перехідник з USB на microUSB**.**

Виводи мікроконтролера:

PA9 -> USB_OTG_VBUS
PA10 -> USB_OTG_ID
PA11 -> USB_OTG_DM
PA12 -> USB_OTG_DP
PC0 -> USB_VBUS_Enable

Необхідні бібліотеки:

Модулі, що підключаються CooCox-IDE: GPIO, MISC

Підтримувані бібліотеки: STM32_UB_FATFS (USB-Version)

Перечислення

```
typedef enum {
  USB_MSC_HOST_NO_INIT =0,   // інтерфейс USB не ініціалізований
  USB_MSC_DEV_DETACHED,      // немає підключеного пристрою
  USB_MSC_SPEED_ERROR,       // Швидкість USB-пристрою не підтримується
  USB_MSC_DEV_NOT_SUPPORTED, // Пристрій не підтримується
  USB_MSC_DEV_WRITE_PROTECT, // Пристрій захищений від запису
```

```
6       USB_MSC_OVER_CURRENT,     // Перевантаження по струму
7       USB_MSC_DEV_CONNECTED     // Пристрій підключений і готовий
8     }USB_MSC_HOST_STATUS_t;
```

Функції:

```
1   void UB_USB_MSC_HOST_Init(void);              // ініціалізувати USB-Host
2   USB_MSC_HOST_STATUS_t UB_USB_MSC_HOST_Do(void);  // перевірити стан пристрою
```

Приклад:

```
1   //--------------------------------------------------------------
2   // File    : main.c
3   // Datum   : 13.04.2013
4   // Version : 1.0
5   // Autor   : UB
6   // EMail   : mc-4u(@)t-online.de
7   // Web     : www.mikrocontroller-4u.de
8   // CPU     : STM32F4
9   // IDE     : CooCox CoIDE 1.7.0
10  // Module  : CMSIS_BOOT, M4_CMSIS_CORE
11  // Funktion : Demo der USB-MSC-HOST-Library
12  // Hinweis  : Diese zwei Files muessen auf 8MHz stehen
13  //            "cmsis_boot/stm32f4xx.h"
14  //            "cmsis_boot/system_stm32f4xx.c"
15  //--------------------------------------------------------------
16
17  #include "main.h"
18  #include "stm32_ub_led.h"
19  #include "stm32_ub_usb_msc_host.h"
20
21  int main(void)
22  {
```

```
23      FIL myFile;   // обробник файлів
24      uint8_t write_ok=0;
25
26      SystemInit(); // Ініціалізація налаштувань кварцу
27
28      // ініціалізація світлодіодів
29      UB_Led_Init();
30
31      // ініціалізація USB-OTG-порту як MSC-HOST
32      // (для читання / запису на USB флешку)
33      UB_USB_MSC_HOST_Init();
34
35      while(1)
36      {
37        // Дані про статус USB
38        if(UB_USB_MSC_HOST_Do()==USB_MSC_DEV_CONNECTED) {
39          // якщо USB-флешка виявлена
40          UB_Led_On(LED_GREEN);
41
42          // Якщо файл ще не записаний
43          if(write_ok==0) {
44            write_ok=1;
45            UB_Led_On(LED_RED);
46            // монтувати флешку
47            if(UB_Fatfs_Mount(USB_0)==FATFS_OK) {
48              // Редагувати файл, записаний в корінь флешки
49              if(UB_Fatfs_OpenFile(&myFile, "USB_File.txt", F_WR_CLEAR)==FATFS_OK) {
50                // Написати кілька рядків тексту в файлі
51                UB_Fatfs_WriteString(&myFile,"Test der WriteString-Funktion");
52                UB_Fatfs_WriteString(&myFile,"hier Zeile zwei");
53                UB_Fatfs_WriteString(&myFile,"ENDE");
54                // Закрити файл
55                UB_Fatfs_CloseFile(&myFile);
56              }
57              // розмонтувати флешку
```

```
58              UB_Fatfs_UnMount(USB_0);
59            }
60            UB_Led_Off(LED_RED);
61          }
62        }
63        else {
64          // Якщо немає доступних USB флешек
65          UB_Led_Off(LED_GREEN);
66        }
      }
    }
```

Завдання на СРС. Охарактеризуйте бібліотечні функції з інтерфейсом USB

Практичне заняття 8. Інтерфейс CAN

Мета Здобуття навичок роботи з блоком комунікації CAN

Завдання Здійснити передачу даних по інтерфейсу CAN . Для перевірки правильності коду забезпечити індикацію на світлодіодах (В залежності від того який байт подано, загориться певна кількість діодів)

Порядок виконання

1.З початку потрібно виставити піни у програмі CubeMx та прописати тактову частоту (рис. 1)

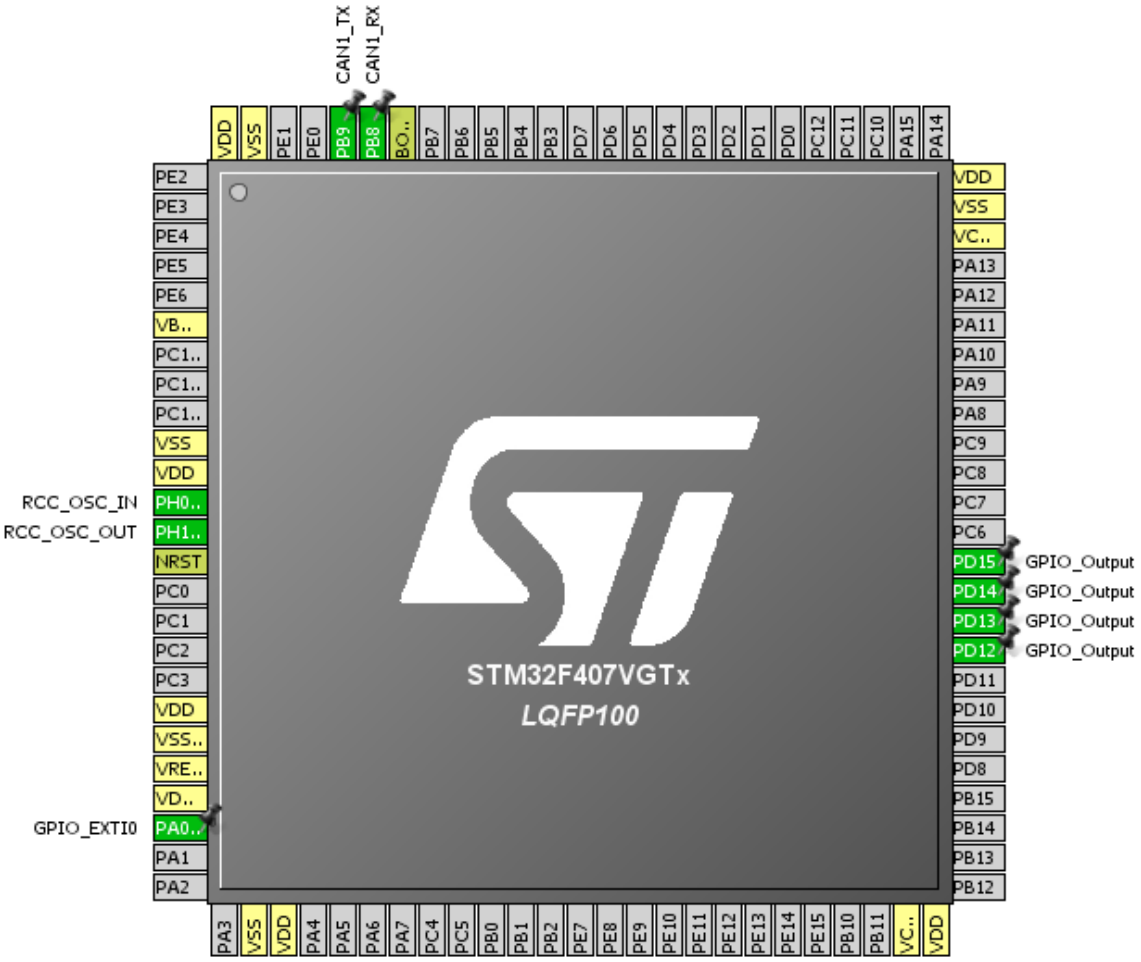

Рис.1. Виводи МК для CAN інтерфейса

Конфігурацію проекту наведено на рис.2.

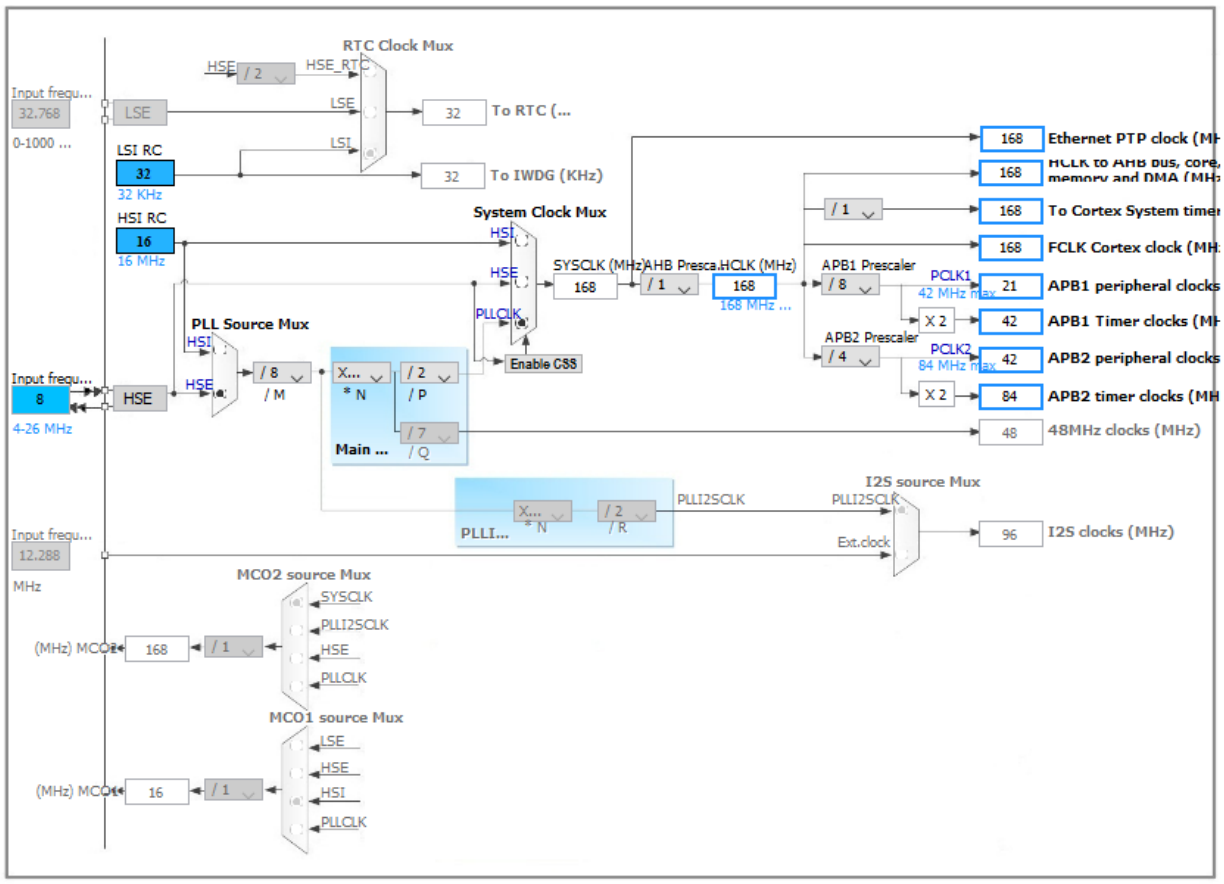

Рис. 2 . Конфігурація проекту

2. Після цього необхідно використати наступній програмний код у середовищі Keil.

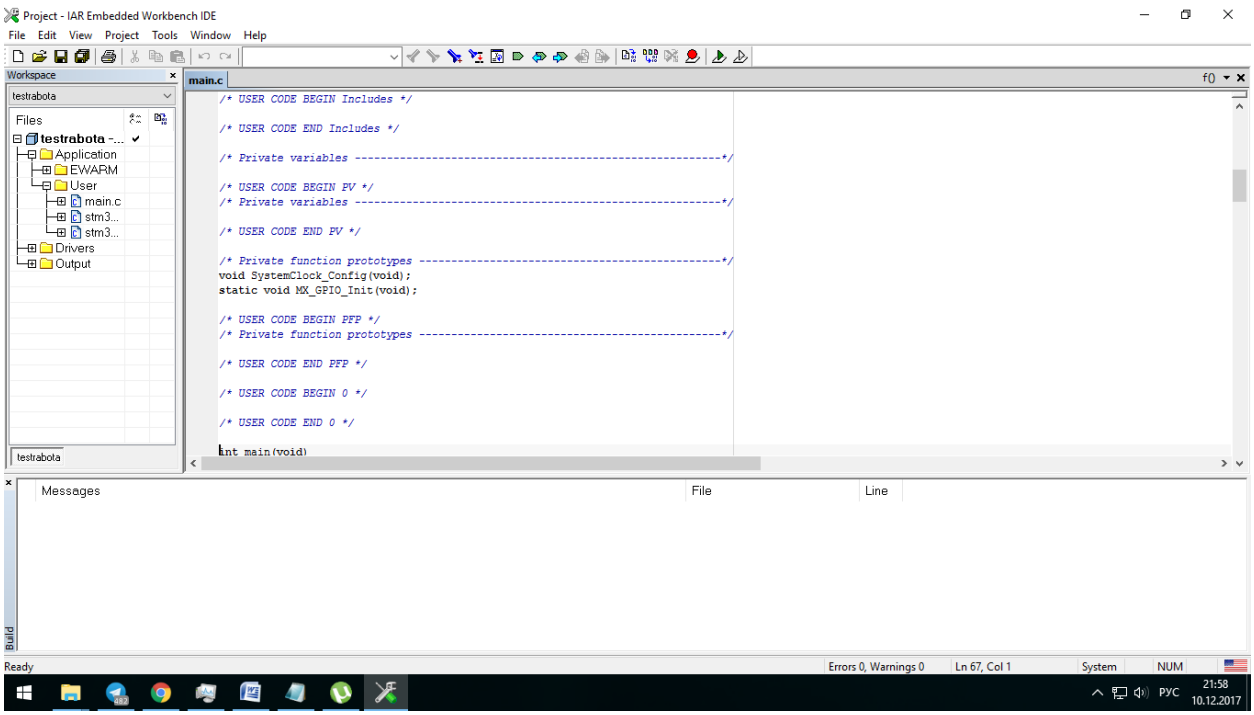

Рис. 3 -Програмний код у середовищі Keil

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
CanTxMsgTypeDef        TxM;
CanRxMsgTypeDef        RxM;
CAN_FilterConfTypeDef sFilterConfig;
uint8_t a;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();

  /* USER CODE BEGIN 2 */
hcan1.pTxMsg=&TxM;
```

```
hcan1.pRxMsg=&RxM;

sFilterConfig.FilterNumber = 0;
sFilterConfig.FilterMode   = CAN_FILTERMODE_IDLIST;
sFilterConfig.FilterScale  = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x244<<5;  // 0x245 other board
sFilterConfig.FilterIdLow  = 0;
sFilterConfig.FilterMaskIdHigh = 0;
sFilterConfig.FilterMaskIdLow  = 0;
sFilterConfig.FilterFIFOAssignment = 0;
sFilterConfig.BankNumber   = 14;
sFilterConfig.FilterActivation = ENABLE;
HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig);

hcan1.pTxMsg->StdId  = 0x245; // 0x244 other board
hcan1.pTxMsg->RTR    = CAN_RTR_DATA;
hcan1.pTxMsg->IDE    = CAN_ID_STD;
hcan1.pTxMsg->DLC    = 1;

HAL_CAN_Receive_IT(&hcan1, CAN_FIFO0);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  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 = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
```

```
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV8;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV4;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* CAN1 init function */
static void MX_CAN1_Init(void)
{

  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 16;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SJW = CAN_SJW_1TQ;
  hcan1.Init.BS1 = CAN_BS1_3TQ;
  hcan1.Init.BS2 = CAN_BS2_5TQ;
  hcan1.Init.TTCM = DISABLE;
  hcan1.Init.ABOM = DISABLE;
  hcan1.Init.AWUM = DISABLE;
  hcan1.Init.NART = DISABLE;
  hcan1.Init.RFLM = DISABLE;
  hcan1.Init.TXFP = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;
```

```
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PD12 PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 1);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
```

```
  /* USER CODE END 6 */

}

#endif
```

3.Для перевірки правильності коду необхідно подавати певний байт. В залежності від того який байт подано, загориться певна кількість діодів.

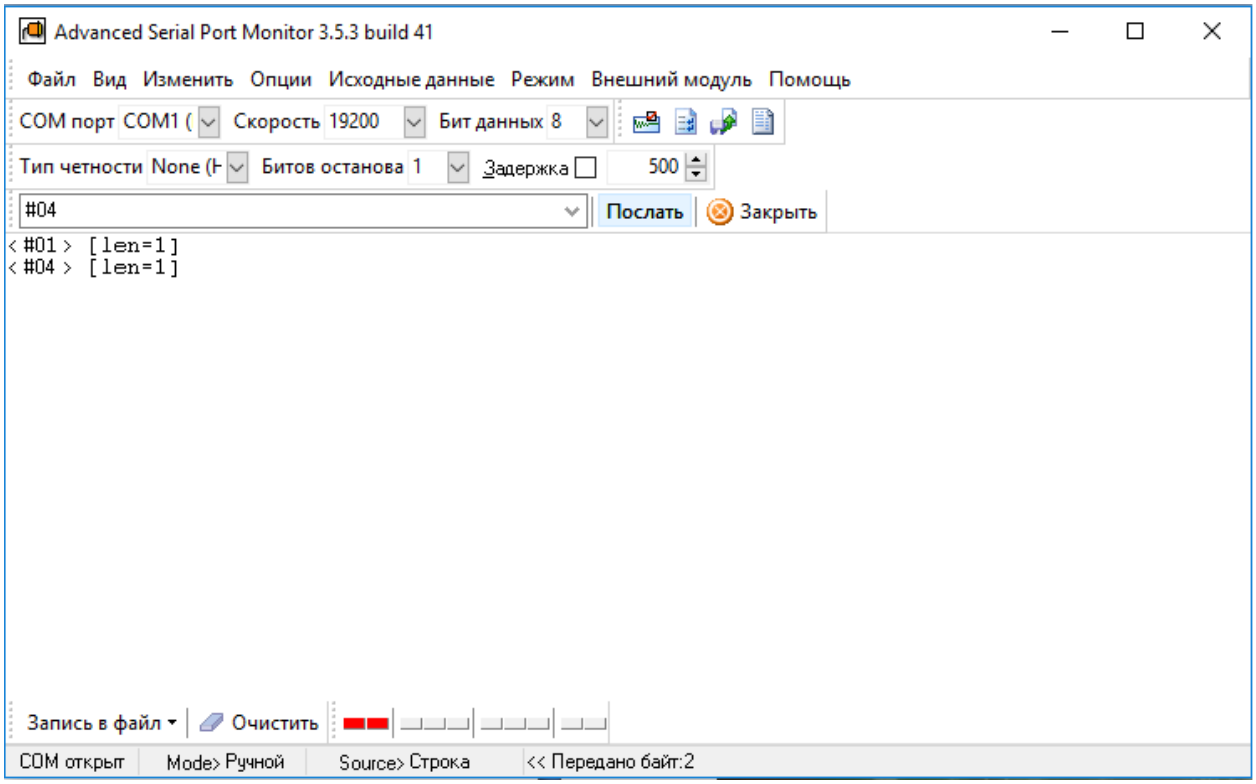

Рис. 4 Пересилання байта перевірки

На рис. 5 і 6 показано результати перевірки для кода 01 (горить один світлодіод) та кода 04 (горять 4 світлодіода)

VERY

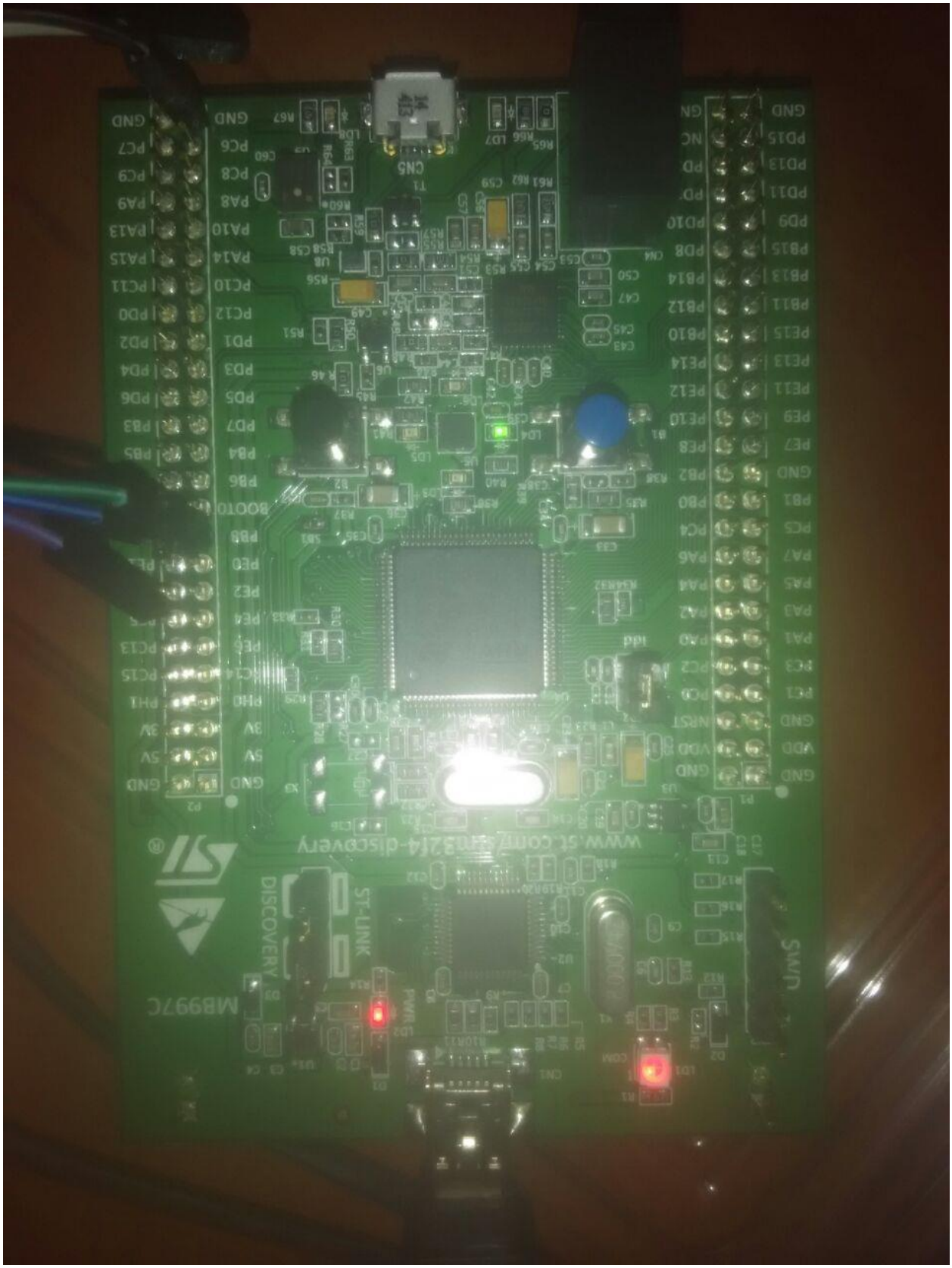

Рис. 5 Результат перевірки для кода 01 (Вигляд плати DISCOVERY)

Рис. 6. Результат перевірки для кода 04 (Вигляд плати DISCOVERY)

Завдання на СРС. Охарактеризуйте бібліотечні функції з інтерфейсом CAN

НАВЧАЛЬНО-МЕТОДИЧНІ МАТЕРІАЛИ

1. Дистанційний курс Сучасні напрямки комп'ютерної та мікропроцесорної техніки Сертифікат УЦДО від 25.04.2013; № НМП №3670 Режим доступу до ресурсу: http://moodle.ipo.kpi.ua/moodle/course/view.php?id=516

2. Дистанційний курс Спеціалізовані та промислові мікропроцесорні системи; Сертифікат УЦДО від 15.05.2012; № НМП №2536 Режим доступу до ресурсу: http://moodle.udec.ntu-kpi.kiev.ua/moodle/course/view.php?id=309

3. Конспект лекцій з дисципліни «Сучасні напрямки комп'ютерної та мікропроцесорної техніки. Розділ 3. Архітектура сучасних мікроконтролерів» для спеціальності 6.050802 – «Електронні пристрої та системи» (171 Електроніка)/ Укладачі: Терещенко Т.О., Ямненко Ю.С., Хохлов Ю.В.: НТУУ «КПІ ім. І. Сікорського», 2015. -230 с. Гриф «Рекомендовано» надано Вченою радою факультету електроніки НТУУ «КПІ»

4. Терещенко Т. О., Тодоренко В.А., Батрак Л.М.,Ямненко Ю. С. Мікропроцесорні пристрої. Навчальний посібник для аспірантів спеціальності «Електроніка». - К.: НТУУ «КПІ ім. Ігоря Сікорського», 2017. - 244с. Гриф надано Вченою радою КПІ ім.Ігоря Сікорського, протокол №6 від 12.06.2017 р.

5. Жуйков В.Я., Терещенко Т.О., Петергеря Ю.С. Електронний підручник «Мікропроцесори і мікроконтролери» - 2009 Гриф надано Міністерством освіти і науки України (лист № 1.4_18-Г-114 від 10.01.2009 р. - режим доступу до ресурсу:. http://www.kaf-pe.ntu-kpi.kiev.ua

6. UART и USART. COM-порт. Часть 1. - режим доступу до ресурсу: http://www.rotr.info/electronics/mcu/arm_usart.htm

7. Описание шины I2C - режим доступу до ресурсу: http://www.itt-ltd.com/reference/ref_i2c.html

8. Описание шины CAN- - режим доступу до ресурсу:: http://itt-ltd.com/reference/ref_can.html

9. Принцип действия шины TWI. режим доступу до ресурсу http://www.gaw.ru/html.cgi/txt/doc/micros/avr/arh_xmega_a/19_3.htm

10. 1-Wire-интерфейс - режим доступу до ресурсу: http://www.elin.ru/1-Wire/

11. Universal serial bus режим доступу до ресурсу: http://www.usb.org

12. Последовательный интерфейс SPI (3-wire) - режим доступу до ресурсу: http://www.gaw.ru/html.cgi/txt/interface/spi/index.htm

13. Дитрих Д., Артемов Н.И., Низамутдинов О.Б., Белковский С.В. Fieldbus-концепция построения систем промышленной автоматизации // Приборы и системы. Управление, Контроль, Диагностика, 11/2000. – С. 35-38.

14. Белковский С.В. Анализ протокола в системах полевых шин // Теоретические и прикладные аспекты информационных технологий: Сб. науч. тр. / НИИУМС. – Пермь, 1999. – Вып. 48. – С. 136-138.