МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
ПРОГРАМНІ ЗАСОБИ ІНТЕРНЕТУ РЕЧЕЙ

Навчальний посібник

Рекомендувала Науково-методична рада
Національного університету “Львівська політехніка”

Львів
Видавництво Львівської політехніки
2021

УДК 004
Г 79

Рецензенти:

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

Шаров Б. Г., кандидат технічних наук, ТзОВ "ЕЛСІ";

Пуйда В. Я., кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин

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

Гребеняк А. В.

Г 79 Лабораторний практикум з дисципліни "Програмні засоби інтернету речей" : навч. посібник / М. В. Ногаль. – Львів: Видавництво Львівської політехніки, 2021. – 48 с. Режим доступу: http://eom.lp.edu.ua/textbooks/np-pzir.pdf вільний. – Заголовок з екрана.

ISBN 978-966-941-688-9

До практикуму увійшли методичні вказівки до лабораторних робіт з навчальної дисципліни "Програмні засоби інтернету речей".

УДК 004

© Гребеняк А. В., 2021
© Національний університет
"Львівська політехніка", 2021

ISBN 978-966-941-688-9

Лабораторна робота № 1

ОЗНАЙОМЛЕННЯ З СЕРЕДОВИЩЕМ WICED-STUDIO

Мета роботи: ознайомитися з середовищем WICED-STUDIO. Набути практичних навичок для користування цим середовищем. Ознайомитись з можливостями плати відлагодження CYW943907AEVAL1F.

1. Ознайомлення з середовищем WICED-STUDIO

Перейдемо на сайт Cypress для отримання останньої версії середовища WICED-STUDIO. Середовище розташоване за посиланням https://www.cypress.com/products/wiced-software.

Для скачування потрібно пройти просту реєстрацію та завантажити середовище WICED-STUDIO.

Також потрібно завантажити додатково файли з проєктами та довідковими матеріалами всього курсу лабораторних робіт. Вони знаходяться за посиланням https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.

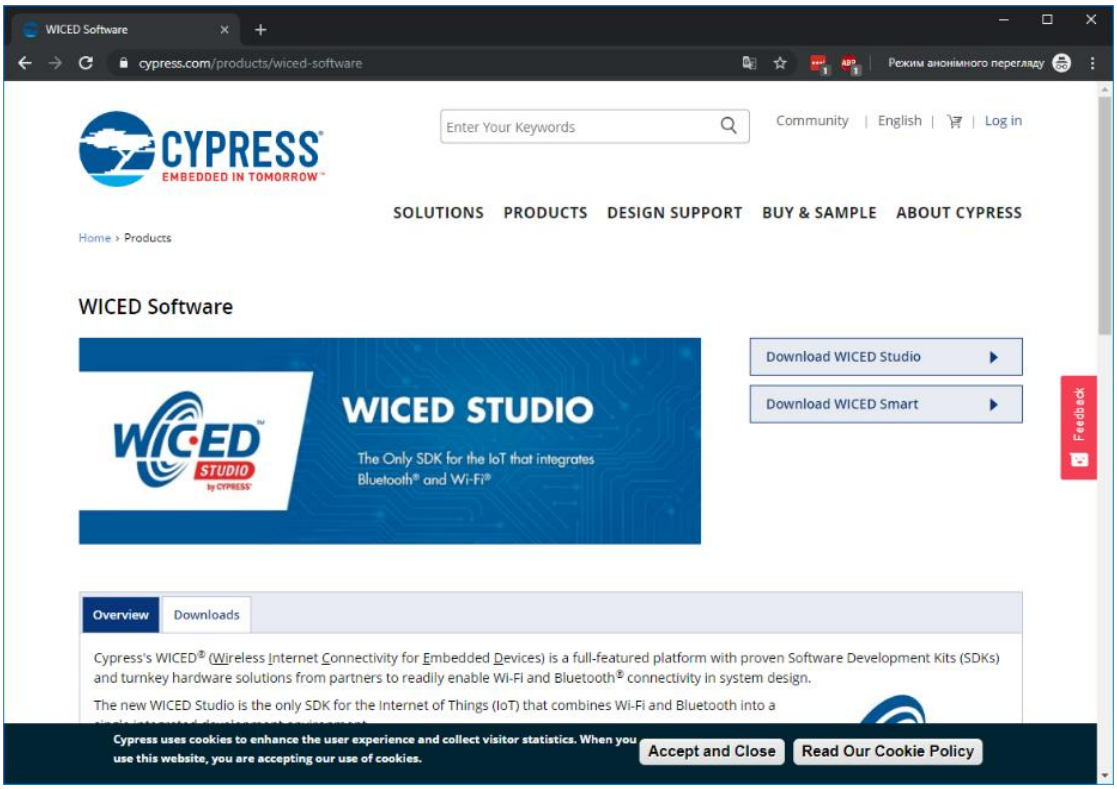


Рис. 1. Вікно ресурсу завантаження середовища

Програмний інструмент WICED називається "WICED-Studio" і заснований на Eclipse. Середовище розробки WICED-Studio встановлюється за замовчуванням в каталог: *C:/Users/<UserName>/AppData/Local/WICED*. Як частина інсталяції WICED-Studio, за замовчуванням створюється робоча область SDK за шляхом *C:/Users/<UserName>/*

Documents/WICED-Studio-<version>/43xxx_Wi-Fi. Робоча область SDK – це місце, де ви будете створювати свої проєкти. Зауважте, що для кожної інсталяції версії WICED-Studio створюється новий набір файлів робочої області SDK. Якщо ви встановите новішу версію WICED-Studio, ваші проєкти з попередньої версії все ще будуть доступні в розташуванні робочого простору SDK, пов'язаному з попередньою версією WICED-Studio. Ви повинні скопіювати їх вручну, якщо ви хочете отримати доступ до них у новій версії.

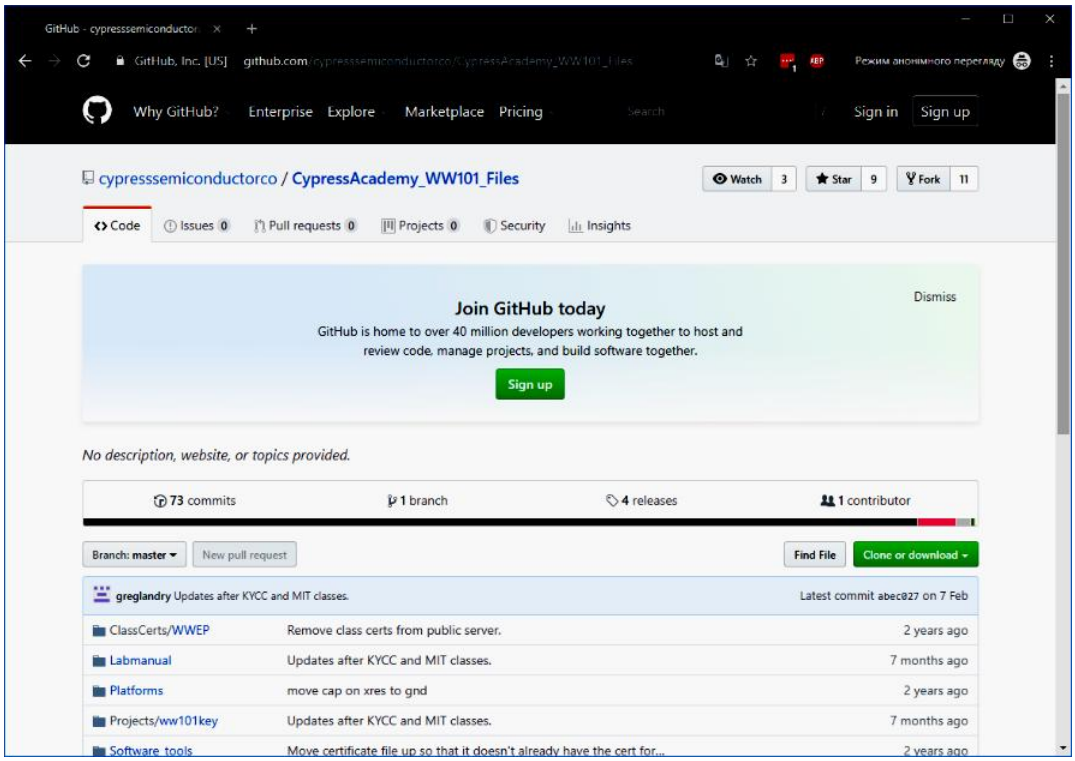


Рис. 2. Вікно ресурсу завантаження проєктів

Після встановлення WICED-Studio відобразиться в Windows у розділі Start > All Programs > Cypress > WICED-Studio. Перший раз, коли ви відкриєте WICED-Studio, вас запитають, яку платформу ви хочете використовувати. Ми будемо використовувати 43xxx_Wi-Fi для цього класу, але якщо ви використовували іншу платформу, не хвилюйтеся – ви можете легко змінити її всередині інструменту, використовуючи випадаюче меню.

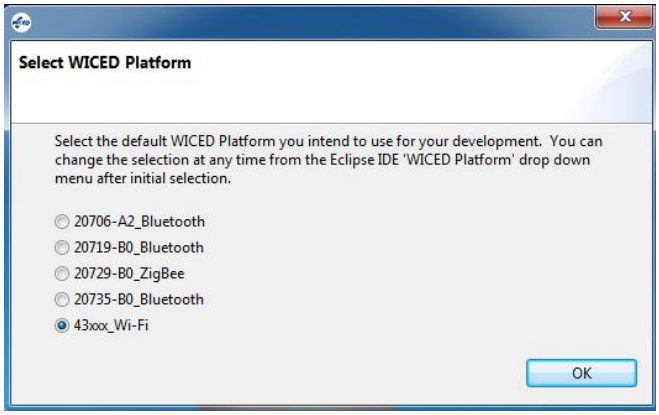


Рис. 3. Вибір платформи

Всі проєкти, які будуть створюватись, розташовані в робочій області SDK. В щойно скачаних файлах є папка з проєктами *ww101key*, яку потрібно перемістити в робочу область за шляхом *C:/Users/<UserName>/Documents/WICED-Studio-<version>/43xxx_Wi-Fi/apps*. Запускаємо середовище WICED.

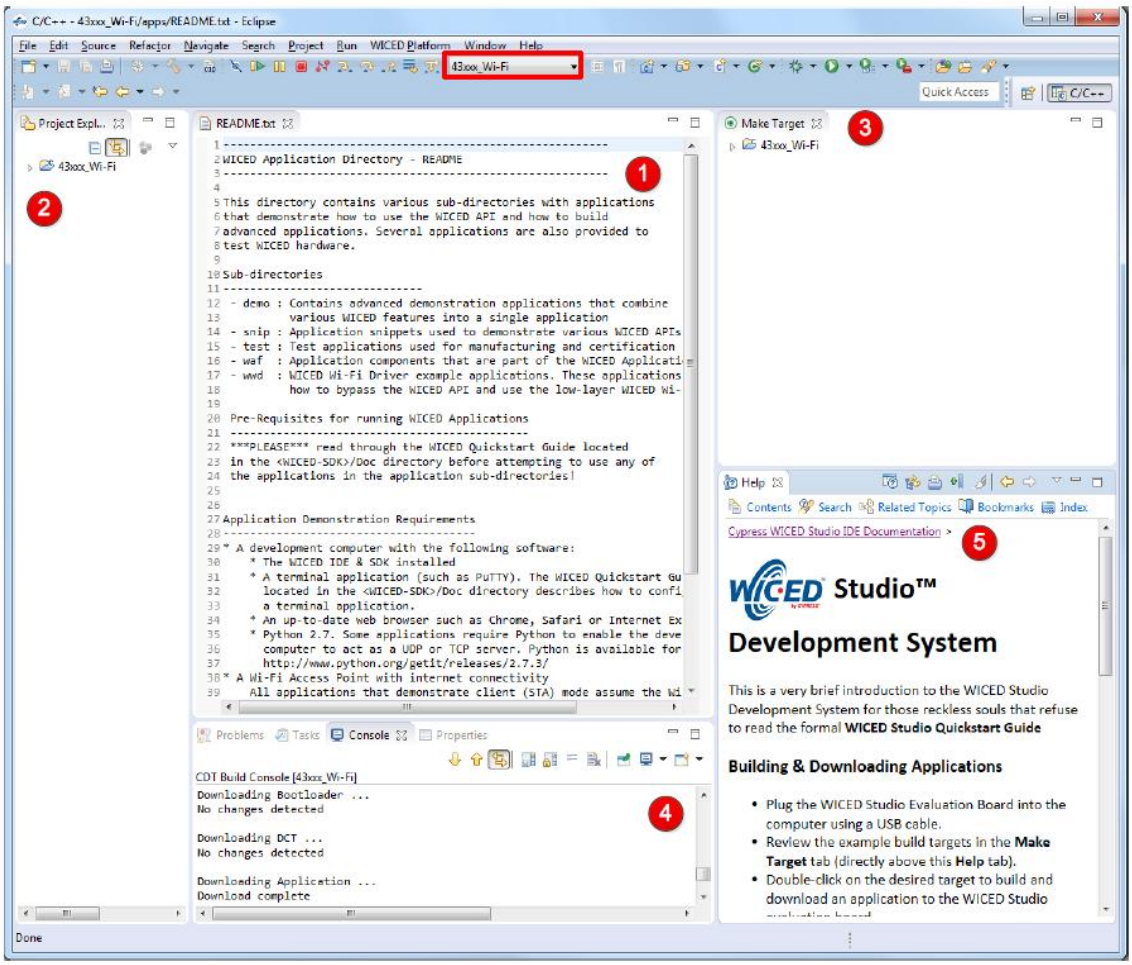


Рис. 4. Вікно середовища WICED-Studio

Основні вікна:

1) File Editor;
2) Project Explorer;
3) Make Target;
4) Console;
5) Help.

Якщо ви ненавмисно закриєте вікно, його можна знову відкрити в меню Вікно> Показати перегляд. Деякі з представлень представлені у розділі Вікно> Показати перегляд> Інше ... Ви можете перетягувати вікна та змінювати їхні розміри за своїм бажанням.

У вікні Project Explorer основними папками є:

- **Apps** – У папці додатків розміщуються всі приклади проєктів, а також де ви розміщуєте власні проєкти. Робоча область SDK включає безліч прикладних проєктів. Вони розбиваються на категорії за назвою папки.

- **Doc** – містить документацію для робочої області SDK. Особливий інтерес викликає файл API.html, який документує всі функції WICED API. Використовувати цей файл, як правило, простіше, якщо ви відкриєте його на вебпереглядачі за власним вибором, а не з WICED-Studio. Ви можете зробити це з WICED-Studio, клацнувши правою кнопкою миші на API.html та обравши "Open With > System Editor". Залежно від вашого веббраузера та налаштувань, можливо, вам доведеться вказати йому дозволити елементам управління ActiveX бачити меню – перше вікно, яке ви побачите, коли відкриєте файл API.html. Ви можете ввести рядки пошуку в поле у верхньому правому куті. Список буде динамічно фільтруватися під час введення. Наприклад, якщо ви введете "wiced_gpio", ви побачите перелік усіх WICED API, які використовуються для контролю вводу–виводу. Іноді функція пошуку перестає працювати. Якщо це трапиться, закрийте сторінку веб-переглядача та відкрийте її знову.
- **Platforms** – містить інформацію про різні набори (тобто апаратні платформи). Ці файли необхідні для програмування певного проєкту на конкретне обладнання. У нашому випадку комплект, який ми використовуємо, називається CYW943907AEVAL1F. Цей комплект має папку платформи, але до нього ми будемо використовувати набір файлів платформи, який також включає периферійні пристрої на екрані.
- **Libraries –** містить різні набори файлів функцій бібліотеки. Наприклад, є бібліотеки для роботи з файловими системами (у папці файлових систем), для використання графічних дисплеїв U8G (у графічній папці) та для читання JSON (у папці утиліт).
- **Resources** – це місце, де ви зберігаєте файли, необхідні вашій програмі. Наприклад, якщо ваша програма містить веб-сервер, HTML-файли для цього сервера знаходяться у папці ресурсів у програмі/https_server. Як інший приклад, сертифікати безпеки також містяться у папці ресурсів.

2. Ознайомлення з платою відлагодження Cypress CYW943907AEVAL1F

- Dual band 2.4 and 5GHz Wi-fi, 1×1 11n
- Ethernet
- SOC w/ ARM CR4 320Mhz
- 2MB on chip RAM
- Secure OTP and HW crypto engine
- USB JTAG Programmer/Debugger

3. Запуск проєкту

Щоб завантажити проєкт на плату відлагодження, вам потрібно буде створити нову цільову форму у вигляді: *<folder1>.[<folder2>...].<project>-<platform> download run*.

Наприклад, якщо ми створимо папку під назвою "ww101" для проєктів нашого класу та підпапку під назвою "02", а перший проєкт називаємо "02_blinkled", ціль збірки для нашої плати (якщо припустимо, що ми використовуємо CYW943907AEVAl1F) буде:

ww101.02.02_blinkled-WW101_2_CYW943907AEVAL1F download run.

Визначення цілей, які вже визначені, можна побачити у вікні "Make Target " уздовж правої частини WICED-Studio. Розгорніть "43xxx_Wi-Fi", щоб побачити існуючі цілі створення.

Щоб утворити нову ціль створення, можна клацнути правою кнопкою миші на існуючій цілі створення, яка схожа на ту, яку ви хочете створити, і вибрати «Нове…». Це дасть вам копію створення цілі з "Копіювати" на початку імені. Видаліть "Копію" (не забудьте видалити пробіл!) та змініть ім’я, як потрібно для нової цілі створення.

Рис. 5. Вигляд плати відлагодження

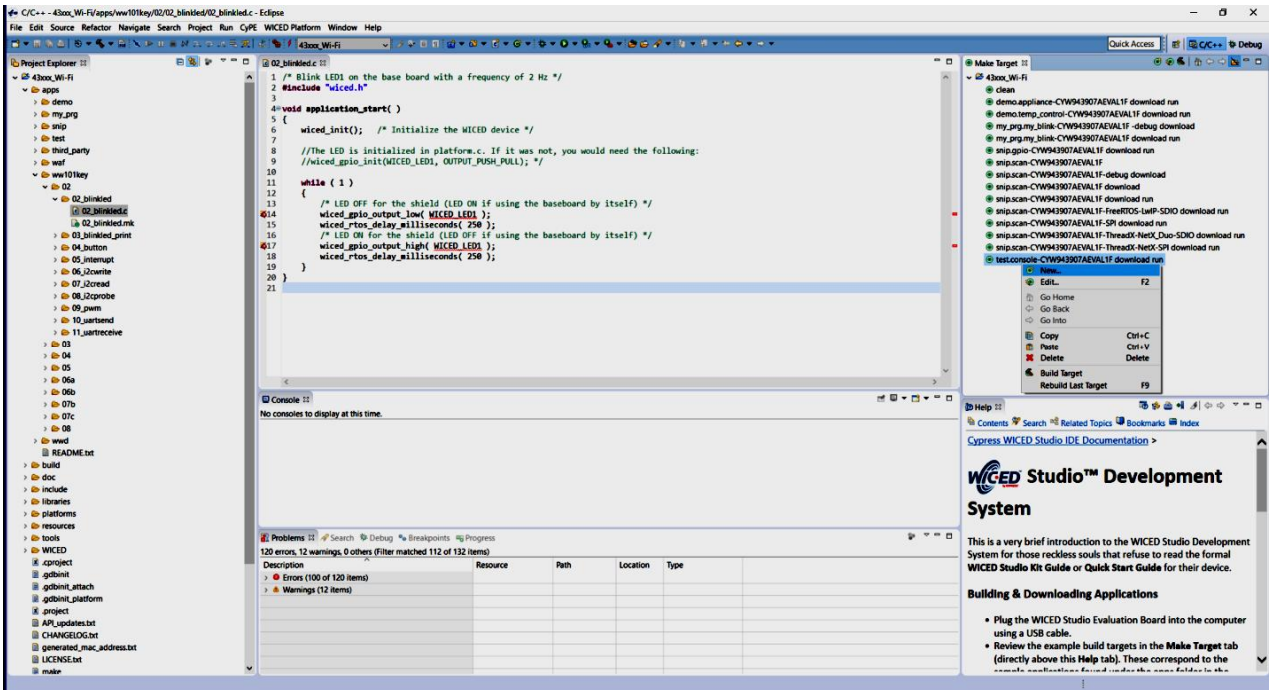


Рис. 6. Створення файлу виконання

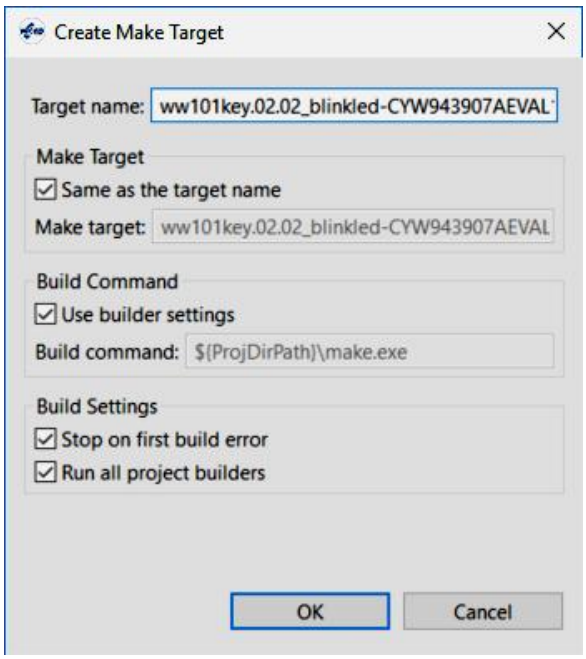


Рис. 7. Внесення імені проєкту

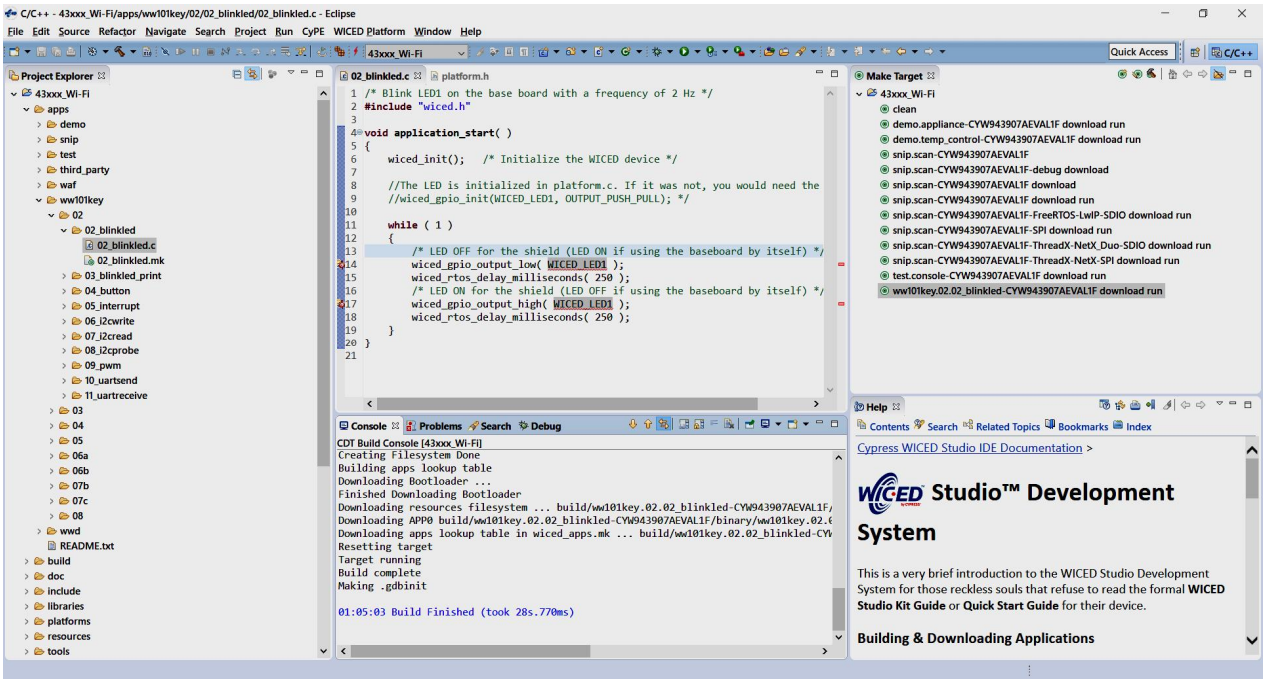


Рис. 8. Вигляд середовища після компіляції проєкту

4. Порядок роботи

1.Встановити середовище WICED-Studio. Додати до середовища також додаткові файли з проєктами.

2.Ознайомитись з будовою відлагоджувальної плати CYW943907AEVAL1F.

3.Під’єднати відлагоджувальну плату до комп’ютера, та запустити середовище WICED-Studio.

4.Створити файл запуску тестового проєкту.

5.Запустити існуючий проєкт на компіляцію на завантаження.

5. Зміст звіту

1.Записати назву та мету виконання лабораторної роботи.

2.Описати хід виконання лабораторної роботи.

3.Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.

4.У висновку оцінити зручність створення на запуску проєктів у середовищі розробки.

6. Контрольні питання

1.Перелічити основні вікна середовища розробки WICED-Studio.

2.Навести перелік пунктів у вікні “Project Explorer”

3.Яку роль виконує кожен з пунктів у вікні “Project Explorer”?

4.Як викликати довідку для платформи CYW943907AEVAL1F?

5.Що необхідно виконати для компіляції та запуску проєкту?

7. Використані джерела

1.Середовище – https://www.cypress.com/products/wiced-software.

2.Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.

3.Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.

4.Плата відлагодження – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

Лабораторна робота № 2

ВИКОРИСТАННЯ WICED-SDK ДЛЯ РОБОТИ З ПЕРИФЕРІЄЮ (GPIO, UART, LED, I2C)

Мета роботи: ознайомитися з WICED-SDK. Набути практичних навичок з користування даними WICED-SDK. Дізнатись, як використовувати периферію в платі відлагодження CYW943907AEVAL1F.

1. Файли платформи (CYW943907AEVAL1F)

Два ключових файли тут platform.c і platform.h. Файл platform.h містить визначення #define і визначення типів, які використовуються для налаштування та доступу до різних периферійних пристроїв набору та екрана. Наприклад, шилд містить два світлодіоди та дві механічні кнопки. Вони ідентифікуються в platform.h, використовуючи імена WICED_LED1, WICED_LED2, WICED_BUTTON1 та WICED_BUTTON2.

Імена, які використовуються, перевизначені з базової плати, тож замість світлодіодів та кнопок на базовій платі ми будемо використовувати відповідні ресурси із шилда. Імена залишаться однаковими – лише платформа, на яку ми орієнтуємось, буде іншою. Це означає, що якщо у вас є проєкт, який використовує світлодіоди та кнопки, ви можете використовувати той самий файл проєкту C та make файл, щоб запустити його, використовуючи лише основну плату або основну плату та екран, просто змінивши ім’я платформи в Make Target!

```
382 /* LEDs and buttons on the shield */
383 #define WICED_LED1       ( WICED_GPIO_12 )
384 #define WICED_LED2       ( WICED_GPIO_6  )
385 #define WICED_BUTTON1    ( WICED_GPIO_10 )
386 #define WICED_BUTTON2    ( WICED_GPIO_8  )
```

Файл platform.c містить кілька постійних масивів та структур, які використовуються для налаштування периферійних пристроїв. Цей файл також містить функції, які використовуються для ініціалізації та управління периферійними пристроями. Наприклад, світлодіодні піни ініціалізуються як виходи, а піни кнопки ініціалізуються як входи з резистивним підтягуванням.

У platform.h ви також знайдете список дійсних периферійних пристроїв. Наприклад, є 6 ШІМ:

```
typedef enum
{
    WICED_PWM_1,
    WICED_PWM_2,
    WICED_PWM_3,
    WICED_PWM_4,
    WICED_PWM_5,
    WICED_PWM_6,
    WICED_PWM_MAX, /* Denotes the total number of PWM port aliases. Not a valid PWM alias */
} wiced_pwm_t;
```

Піни, що використовуються для кожної ШІМ, можна знайти на platform.c.

```
/* PWM peripherals. Used by WICED/platform/MCU/wiced_platform_common.c */
const platform_pwm_t platform_pwm_peripherals[] =
{
    [WICED_PWM_1]  = {PIN_GPIO_10,  PIN_FUNCTION_PWM0, },   /* or PIN_GPIO_0, PIN_GPIO_8,  PIN_GPIO_12, PIN_GPIO_14, PIN_GPIO_16, PIN_PWM_0
    [WICED_PWM_2]  = {PIN_GPIO_11,  PIN_FUNCTION_PWM1, },   /* or PIN_GPIO_1, PIN_GPIO_7,  PIN_GPIO_9,  PIN_GPIO_13, PIN_GPIO_15, PIN_PWM_1
    [WICED_PWM_3]  = {PIN_GPIO_16,  PIN_FUNCTION_PWM2, },   /* or PIN_GPIO_8, PIN_GPIO_0,  PIN_GPIO_10, PIN_GPIO_12, PIN_GPIO_14, PIN_PWM_2
    [WICED_PWM_4]  = {PIN_GPIO_15,  PIN_FUNCTION_PWM3, },   /* or PIN_GPIO_1, PIN_GPIO_7,  PIN_GPIO_9,  PIN_GPIO_11, PIN_GPIO_13, PIN_PWM_3
    [WICED_PWM_5]  = {PIN_PWM_4,    PIN_FUNCTION_PWM4, },   /* or PIN_GPIO_0, PIN_GPIO_8,  PIN_GPIO_10, PIN_GPIO_12, PIN_GPIO_14, PIN_GPIO_16
    [WICED_PWM_6]  = {PIN_GPIO_7,   PIN_FUNCTION_PWM5, },   /* or PIN_GPIO_1, PIN_GPIO_9,  PIN_GPIO_11, PIN_GPIO_13, PIN_GPIO_15, PIN_PWM_5
};
```

Зауважте, що імена ШІМ повинні використовуватися у викликах функції PWM API. Тобто, ви повинні використовувати WICED_PWM_1 для використання ШІМ 1. Ви не можете використовувати PIN_GPIO_10 у викликах функції PWM API.

2. Документація

Документацію можна знайти в папці документів SDK Workspace. Файл API.html містить документацію API, яку ми будемо використовувати. Відкрийте цей файл, клацнувши його правою кнопкою миші та виберіть "*Open With > System Editor*", а потім розгорніть "Components" та "Platform Functions", щоб переглянути список підтримуваних компонентів. Ми будемо використовувати GPIO, PWM, UART та I2C.

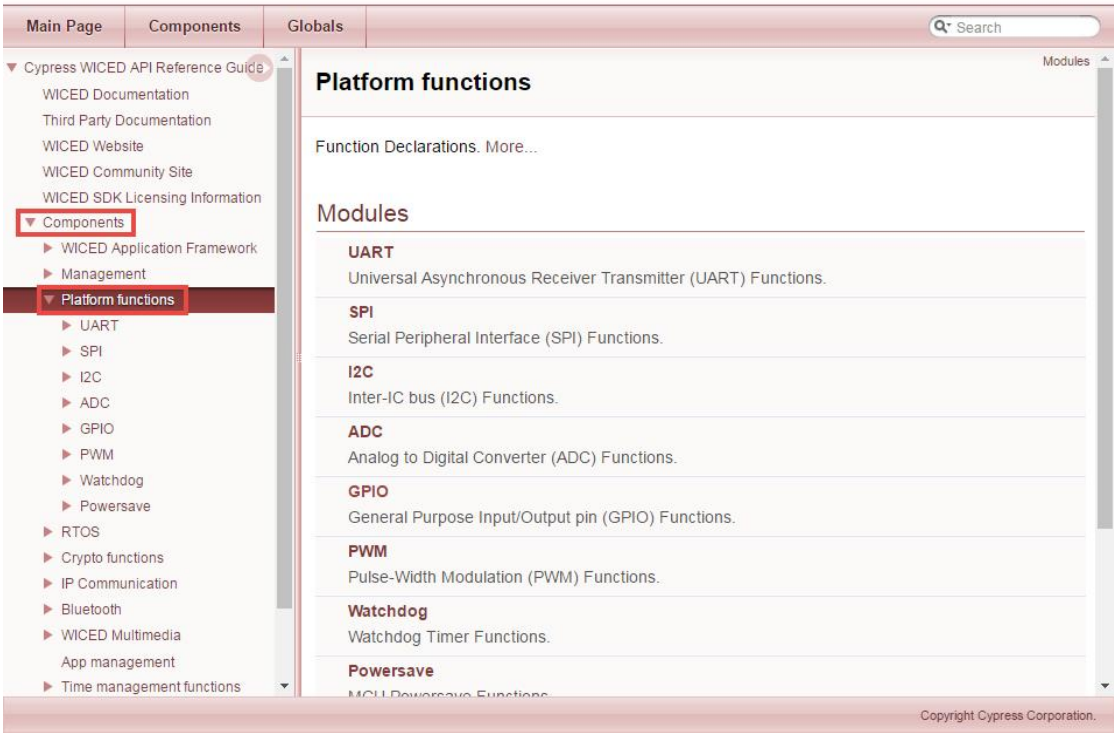


Рис. 1. Вікно довідки

Клацніть на GPIO, щоб переглянути список API GPIO, а потім натисніть функцію wiced_gpio_init для опису. В описі йдеться про те, що функція виконує, але не дає інформації про необхідну структуру конфігурації. Щоб знайти цю інформацію, коли ви перебуваєте в WICED-Studio, ви можете виділити параметр у файлі .c, клацніть правою кнопкою миші та виберіть "Open Declaration". Якщо у вас ще не вказано дійсний параметр, ви також можете потрапити туди, скориставшись "Open Declaration" на ім'я функції, потім тип параметра, а потім ім'я типу. Це покаже вам тип даних із поясненням дозволених варіантів.

3. Периферія

GPIO (Порти вводу–виводу). Як було описано раніше, GPIO повинні бути ініціалізовані перед їх використанням. Піни вводу виводу, які підключені до певної периферії, такої як світлодіоди та кнопки, часто автоматично ініціалізуються для вас як частина файлів платформи. Після ініціалізації вхідні піни можна читати за допомогою wiced_gpio_input_get (), а виходами можна керувати за допомогою wiced_gpio_output_high () та wiced_gpio_output_low (). Параметр цих функцій – це ім'я пінів WICED, наприклад WICED_GPIO_1, або периферійне ім'я для вашої платформи, наприклад WICED_LED1. Переривання GPIO контролюються за допомогою wiced_gpio_input_irq_enable () та wiced_gpio_input_irq_disable ().

PWM (ШІМ). ШІМ має функцію API для вибору, яку ШІМ використовувати, встановлювати частоту (в Гц) та робочий цикл (у відсотках). Ця функція використовується для ініціалізації та зміни частоти або робочого циклу після запуску ШІМ. Він також має функції для запуску та зупинки виводу. Докладніше див. Документацію API.

На додаток до ініціалізації ШІМ необхідно також викликати функцію запуску для введення параметрів у дію та для ШІМ для отримання результату. Вам треба викликати функцію запуску щоразу, коли ви оновлюєте параметри за допомогою функції init.

Якщо ви використовуєте ШІМ на піні, який був ініціалізований як GPIO, такий як світлодіоди на екрані, спочатку потрібно викликати функцію деініціалізації пінів, перш ніж ШІМ зможе вивести сигнал на цей контакт.

UART (асинхронний послідовний порт). На додаток до функцій відлагодження USB-UART, пристрій також може надсилати/приймати стандартні дані UART через піни Arduino UART (D0 і D1), використовуючи STDIO_UART, як визначено у файлі "platform.h". Ці піни також підключені до вбудованого мосту USB-UART, тому те саме вікно терміналу, яке використовується для повідомлень відлагодження, буде працювати і для стандартного зв'язку UART. На наборі CYW943907AEVAL1F є другий UART (WICED_UART_2), підключений до пінів Arduino D8 і D9.

Є функції API для ініціалізації, передачі та прийому UART. Детальну інформацію про ці функції див. У документації API.

Якщо ви використовуєте STDIO_UART, визначений на платформі, вам не потрібно викликати функцію ініціалізації, і не потрібно встановлювати буфер, як описано нижче, оскільки ці функції вже викликаються з platform_stdio_init (), який, своєю чергою, викликається з "platform.c". Вони потрібні лише в тому випадку, якщо ви використовуєте інший інтерфейс UART або інші налаштування UART. STDIO_UART за замовчуванням встановлений для 115200 бод, шириною 8 біт, без паритету, без контролю потоку.

Якщо ви хочете відключити функціональність STDIO_UART або використовувати цей інтерфейс з різними налаштуваннями, додайте до файлу make для проєкту наступне:

GLOBAL_DEFINES + = WICED_DISABLE_STDIO.

Після цього ви більше не побачите стандартну інформацію про час завантаження, що відображається на терміналі.

Для функції ініціалізації UART потрібна структура конфігурації типу wiced_uart_config_t з наступними елементами. Це визначено в "platform_peripheral.h". Як було сказано вище, ви можете знайти цю структуру, виділивши, клацнувши правою кнопкою миші та вибравши "Open Declaration" з WICED-Studio на ім'я функції, тип параметра та ім'я типу.

Якщо ви використовуєте UART для отримання даних, ви повинні створити буфер типу wiced_ring_buffer_t. Цей буфер повинен бути ініціалізований за допомогою функції ring_buffer_init (), яка вимагає вказівника на кільцевий буфер, вказівника на масив для зберігання даних та розміру буфера. Наприклад, для створення 10-байтного буфера кільця під назвою rx_buffer можна використати таке:

#define RX_BUFFER_SIZE (10);
wiced_ring_buffer_t rx_buffer;
uint8_t rx_data [RX_BUFFER_SIZE];
ring_buffer_init (& rx_buffer, rx_data, RX_BUFFER_SIZE);/* Ініціалізувати кільцевий буфер для утримання даних прийому * /.

Після встановлення буфера ви можете читати з UART за допомогою функції wiced_uart_receive_bytes (). Він прочитає вказану кількість байтів з кільцевого буфера для цього інтерфейсу UART і помістить їх у буфер, який ви надаєте. Функція поверне WICED_SUCCESS, якщо байт був отриманий, щоб ви могли сказати, чи був байт у буфері, чи ні.

I2C. Пристрій містить два masters I2C під назвою WICED_I2C_1 та WICED_I2C_2. OLED-дисплей та PSoC на екрані підключаються до WICED_I2C_2.

Як і в інших периферійних пристроях, потрібно ініціалізувати блок за допомогою функції ініціалізації. Однак у цьому випадку параметр, який ви передаєте – це не назва блоку, а структура типу wiced_i2c_device_t. Ця структура містить інформацію про slave I2C, з яким ви збираєтесь спілкуватися. Наприклад, наступне може бути використане для ініціалізації блоку 2 I2C для підключення до slave за адресою 0×08 зі швидкістю 100 кГц (стандартна швидкість).

```
const wiced_i2c_device_t i2cDevice = {
        .port = WICED_I2C_2,
        .address = I2C_ADDRESS,
        .address_width = I2C_ADDRESS_WIDTH_7BIT,
        .speed_mode = I2C_STANDARD_SPEED_MODE
};
```

Існує два способи зчитування/запису даних зі slave. Існує спеціальна функція читання під назвою wiced_i2c_read і спеціальна функція запису під назвою wiced_i2c_write. Існує також функція під назвою wiced_i2c_transfer, яка може робити читання, запис або те і інше. Ми зупинимося на окремих функціях тут, але не соромтеся переглянути функцію передачі в документації та експериментувати з нею. Деякі набори можуть підтримувати лише один із двох методів.

wiced_i2c_read та wiced_i2c_write

Функції читання/запису (wiced_i2c_read та wiced_i2c_write) вимагають вказівника на структуру пристрою, яку було встановлено раніше, набір прапорців, вказівник на буфер для читання/запису та кількість байтів для читання/запису.

Можливі прапори:

WICED_I2C_START_FLAG

WICED_I2C_STOP_FLAG

WICED_I2C_REPEATED_START_FLAG

Як правило, для повної транзакції читання або запису використовуйте прапори "Start" і "Stop" з АБО:

WICED_ I2C_START_FLAG | WICED_I2C_STOP_FLAG

Для буфера ви можете налаштувати структуру для відображення регістрів I2C у slave, до якого ви звертаєтесь. У такому випадку, якщо елементи структури не всі 32-бітні величини, ви повинні використовувати атрибут packed, щоб не 32-бітові величини не були залиті, що призведе до неправильних даних. Наприклад, якщо у вас є байт під назвою "control" з 32-бітовим значенням під назвою "температура", ви можете встановити такий буфер:

```
struct {
        uint8_t control;
        float temperature;
} __attribute__((packed)) buffer;
```

Перед і після слова "attribute" є два підкреслення, а навколо слова "packed " є два набори дужок.

wiced_i2c_transfer

Якщо ви вирішили використовувати функцію wiced_i2c_transfer замість окремих функцій читання/запису, спочатку потрібно встановити структуру повідомлень типу wiced_i2c_message_t. Є три функції, які можна використовувати для цієї мети: wiced_i2c_init_tx_message (), wiced_i2c_init_rx_message (), wiced_i2c_init_combined_message (). Детальну інформацію про ці функції див. у документації API. Параметр "retries" повинен бути встановлений на ненульове значення (наприклад, 1). Значення 0 означає, що навіть не намагайтеся надсилати повідомлення один раз.

Для комплекту CYW943907AEVAL1F I2C не підтримує DMA. Тому параметр "ones_dma" у виклику ініціалізації повідомлення повинен бути встановлений на WICED_TRUE. В іншому випадку передача I2C не вдасться.

Після того, як повідомлення налаштоване, використовуйте wiced_i2c_transfer (), щоб надіслати або отримати повідомлення.

wiced_i2c_probe_device

Ви також можете скористатися wiced_i2c_probe_device (), щоб перевірити, чи є в цій адресі slave I2C. Функція поверне WICED_TRUE, якщо пристрій знайдено, і WICED_FALSE, якщо пристрій не знайдено. Ви все ж повинні ініціалізувати пристрій за допомогою wiced_i2c_init (), перш ніж використовувати wiced_i2c_probe_device ().

4. Порядок роботи

1. Ознайомитись з файлами котрі описують платформу (platform.h, platform.c) а також принциповою схемою плати CYW943907AEVAL1F.
2. Запустити проєкт мигання світлодіодом "02_blinkled". Замінити період затримки, ознайомитись з використаними функціями
3. Запустити проєкт мигання світлодіодом "03_blinkled_print" Запустити термінальну програму (Terminal19b або щось схоже). Замінити повідомлення, ознайомитись з використаними функціями.
4. Запустити проєкт "04_button". Додати моргання діодом в момент натискання. Ознайомитись з використаними функціями.
5. Запустити проєкт "05_interrupt". Змінити режим ініціалізації переривання. Ознайомитись з використаними функціями.

6. Запустити проєкт "06_i2cwrite". Змінити способи перемикання світлодіодів у результаті натискання на кнопки. Ознайомитись з використаними функціями.

7. Запустити проєкт "07_i2cread" Змінити інформацію котра виводиться. Ознайомитись з використаними функціями.

5. Зміст звіту

1. Написати назву та мету виконання лабораторної роботи.
2. Описати хід виконання лабораторної роботи.
3. Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.
4. У висновку оцінити зручність створення на запуску проєктів у середовищі розробки.

6. Контрольні питання

1. У таблиці вгорі platform.h йдеться про те, що WICED_LED1 підключається до WICED_GPIO_12, заголовка D Arduino D5 та WICED_PWM_3. Поясніть, як було визначено це відображення.
2. Чому ви не можете прочитати значення світлодіоду за допомогою функції wiced_gpio_input_get () замість змінної для запам'ятовування стану?
3. У якому файлі та в якому рядку WICED_LED1 присвоюється правильний пін для цієї плати відлагодження?
4. У якому файлі та в якому рядку встановлено пін, підключений до WICED_LED1, як вихід?
5. Скільки портів I2C є на платі відлагодження?

7. Використані джерела

1. Середовище – https://www.cypress.com/products/wiced-software.
2. Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.
3. Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.

Відлагоджувальна плата – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

ЛАБОРАТОРНА РОБОТА № 3

ОПЕРАЦІЙНІ СИСТЕМИ РЕАЛЬНОГО ЧАСУ (RTOS) В СЕРЕДОВИЩІ WICED

Мета роботи: отримати фундаментальне розуміння ролі WICED RTOS у створенні WICED-проєктів. Ознайомитись з використанням шару абстракції WICED RTOS для створення та використання потоків, семафорів, м'ютексів, черг та таймерів.

1. Вступ до систем реального часу (RTOS)

Мета RTOS – зменшити складність написання вбудованої мікропрограми, що має безліч асинхронних завдань, які мають вимоги до ресурсів, що перекриваються. Наприклад, у вас може бути пристрій, який читає і записує дані у підключену мережу, читає і записує дані у зовнішню файлову систему, а також читає і записує дані з периферійних пристроїв. Вирішити вимогу часу відповідати на мережеві запити, продовжуючи підтримувати периферійні пристрої, може бути складною, і тому схильною до помилок задачею. За допомогою RTOS можливо розділити системні функції на окремі завдання (звані потоками) та розробити їх дещо незалежно.

RTOS підтримує список потоків, які стоять у режимі очікування, зупиняються або працюють, і які завдання потрібно виконати наступними (на основі пріоритету) та в який час. Ця функція в RTOS називається планувальником. Існує дві основні схеми управління тим, які потоки/завдання/процеси є активними в операційних системах: попереджувальна та кооперативна.

Під час попереджувальної багатозадачності процесор повністю контролює, яке завдання виконується, і може зупиняти і запускати їх за потребою. У цій схемі планувальник використовує захищені процесором режими, щоб вирвати управління з активних завдань, зупинити їх і перейти до наступного завдання. Вищі завдання з вищими пріоритетами будуть виконуватись перед завданнями з нижчим пріоритетом. Тобто, якщо завдання виконується, і завдання з більш високим пріоритетом вимагає повороту, RTOS призупинить завдання з нижчим пріоритетом і перейде до завдання з вищим пріоритетом. Переважна багатозадачність – це схема, яка використовується у Windows, Linux тощо.

У кооперативній багатозадачності кожен процес повинен бути хорошим мешканцем і повертати ресурси назад до RTOS. Існує низка механізмів управління контролем, наприклад, затримки (rtos_delay_milliseconds), семафори, мютекси та черги (про які ми напишемо згодом тут).

WICED RTOS налаштована в гібридному режимі. Вона переважає для вирішення більш високих пріоритетних завдань, але співпрацює для рівномірних завдань. Тому завдання з більш високим пріоритетом завжди виконуватимуться за рахунок завдань із нижчим пріоритетом, тому важливо забезпечити контроль, щоб дати завданням з нижчим пріоритетом можливість виконатись. Якщо ні, завдання, які не піддаються контролю, взагалі не дозволять виконувати нижчі або однакові пріоритетні завдання. Для запобігання подібним ситуаціям добре застосовувати певну форму механізму контролю ресурсу в кожному потоці.

2. Проблеми з RTOS-системами

Все це звучить чудово, але все не так добре. Є три серйозні помилки, які легко можна створити в таких системах, і ці помилки потім знайти дуже важко. Усі ці помилки викликані побічними ефектами взаємодії між потоками. Велика трійка:

Циклічні залежності, які можуть спричинити тупики.

- Конфлікти з ресурсами при спільному використанні пам'яті та спільному використанні периферійних пристроїв, що може спричинити хаотичну недетерміновану поведінку.
- Труднощі у здійсненні між процесорної комунікації.

Але не все втрачено. WICED RTOS надає вам механізми для вирішення цих проблем, зокрема семафори, мютекси, черги та таймери. Усі ці функції, як правило, працюють однаково. Основний процес роботи такий:

1. Почати зі створення структури даних потрібного типу (наприклад, wiced_mutex_t).

2. Викликати функцію ініціалізації RTOS (наприклад, wiced_rtos_init_mutex ()). Надати його вказівником на структуру, створену на першому кроці. Це "handle", яку використовують інші функції.

3. Отримати доступ до структури даних за допомогою однієї з функцій доступу (наприклад, wiced_rtos_lock_mutex ()).

4. Якщо вона більше не потрібна, звільнити структуру даних відповідною функцією de-init (наприклад, wiced_rtos_deinit_mutex ()).

Усі ці функції повинні мати доступ до структури даних, тому загалом оголошують ці "спільні" ресурси статичними глобальними змінними у файлі, якими вони користуються.

3. Потоки

Як обговорювалося раніше, потоки лежать в основі RTOS. Створити новий потік легко, викликавши функцію wiced_rtos_create_thread () з такими аргументами:

- wiced_thread_t* thread – вказівник на структуру даних обробки потоку. Цей handle використовується для ідентифікації потоку для інших функцій потоку. Спочатку потрібно створити структуру даних handle, перш ніж надати показник функції створення потоку;
- uint8_t priority – це пріоритет потоку.
  - Пріоритети можуть бути від 0 до 31, де 0 є найвищим пріоритетом.
  - Якщо планувальник знає, що два потоки мають право на запуск, він запустить потік з більш високим пріоритетом.
  - WICED Wi-Fi драйвер (WWD) працює з пріоритетом 3. Зазвичай потрібно використовувати нижчий пріоритет, ніж цей.
  - Потік application_start працює з пріоритетом 7. Тому потрібно переконатись що відбувається керування цією функцією, якщо очікується, що будуть запущені потоки з пріоритетом 7 або нижче;
- char *name – ім'я для потоку. Це ім'я використовується лише відлагоджувачем. Ви можете дати йому будь-яке ім'я або просто використовувати NULL, якщо ви не хочете конкретного імені;
- wiced_thread_function_t *thread – вказівник функції на функцію, що є потоком;
- uint32_t stack size – скільки байтів має бути в стеку потоку (потрібно бути обережними, оскільки закінчення стеку може спричинити нестабільну, важку для налаго-

дження поведінку. Використання 10000 є надмірним, але буде працювати для будь-яких задач. Практично будь-який потік потребує більше 100).

- Метод друку максимального розміру стека, який використовується потоком, буде показаний пізніше;

- wiced_thread_arg_t *arg – загальний аргумент, який буде переданий у потік. Це uint32.

- Якщо не потрібно передавати аргумент до потоку, просто використовується NULL.

Наприклад, якщо потрібно створити потік, який виконує функцію "mySpecialThread", ініціалізація може виглядати приблизно так:

```
#define THREAD_PRIORITY      (10)
#define THREAD_STACK_SIZE    (10000)
.
.
wiced_thread_t mySpecialThreadHandle;
.
.
wiced_rtos_create_thread(&mySpecialThreadHandle, THREAD_PRIORITY,
"mySpecialThreadName", mySpecialThread, THREAD_STACK_SIZE, NULL);
```

Функція потоку повинна відповідати типу wiced_thread_function_t. Він повинен приймати єдиний аргумент типу wiced_thread_arg_t і мати недійсне повернення.

Тіло потоку виглядає так само, як "основна" функція application_start програми (насправді ця функція – це лише потік, який автоматично ініціалізується). Часто потік працюватиме завжди (як і основний), тому він матиме розділ ініціалізації та цикл (1), який повторюється завжди. Наприклад:

```
void mySpecialThread(wiced_thread_arg_t arg)
{
    const int delay=100;
    while(1)
    {
        processData();
        wiced_rtos_delay_milliseconds(delay);
    }
}
```

Примітка. Зазвичай варто вставити деяку кількість wiced_rtos_delay_milliseconds() у кожну нитку, щоб інші потоки отримали шанс запуститися. Це стосується основної програми, в циклі while(1) також, оскільки основний додаток – це лише інша нитка. Виняток – якщо у вас є якась інша функція управління потоком, така як семафор або черга, які гарантовано можуть викликати періодичну паузу.

Зауважте, що якщо основний потік програми (application_start) виконує ініціалізацію та запускає інші потоки, то ви можете повністю усунути цикл while(1) з цієї функції. У такому випадку після запуску інших потоків функція application_start просто вийде і не займе більше циклів процесора. Якщо ви це зробите, переконайтеся, що будь-які змінні,

необхідні поза цим потоком (наприклад, handle потоків, handle семафору тощо), оголошуються як глобальні за межами application_start. В іншому випадку вони будуть невизначені після запуску application_start.

Якщо ви хочете побачити, який розмір стеку використовує потік, вам потрібно використовувати основні функції RTOS для RTOS, який ви використовуєте. Наприклад, якщо ви використовуєте ThreadX (RTOS за замовчуванням у WICED), можете додати таку функцію:

```
/* This function will print out the max amount of stack a thread has used */
/* This is ThreadX specific code so it will only work for ThreadX RTOS */
uint32_t maxStackUsage(TX_THREAD *thread)
{
    uint8_t *end =   thread->tx_thread_stack_end;
    uint8_t *start = thread->tx_thread_stack_start;
    while(start < end)
    {
        if(*start != 0xEF)
        {
            return end-start;
        }
        start++;
    }
    return 0;
}
```

Наведена вище функція поверне максимальну кількість байтів, що стек використовувався з моменту її запуску. Ви можете час від часу викликати цю функцію у своєму проєкті (наприклад, у таймері), а потім надрукувати значення. Наприклад:

WPRINT_APP_INFO(("Max Stack: %d\n",(int) maxStackUsage((TX_THREAD*) &mySpecialThreadHandle)));

Функції, доступні для роботи з потоками, знаходяться в розділі "Component→RTOS→Threads" керівництва API.

4. Семафори

Семафор – це механізм сигналізації між потоками. Назва семафор походить від сигналів прапорів корабля. У WICED-SDK семафори реалізовані як просте непідписане ціле число. Коли ви "встановлюєте" семафор, він збільшує значення семафора. Коли ви "отримуєте" семафор, він зменшує значення, але якщо значення дорівнює 0, потік буде відкладати себе, поки семафор не буде встановлений. Можливо використовувати семафор для сигналізації між потоками про те, що щось готово. Наприклад, може бути потік "sendToCloud" та "collectionDataThread". Потік sendToCloud "отримає" семафор, який буде призупиняти потік, доки збирання DataThread "встановлює" семафор, коли у нього є нові дані, які потрібно надіслати до хмари.

Функція get вимагає параметра тайм-ауту. Це дозволяє продовжувати потік через певний проміжок часу, навіть якщо семафор не встановлений. Це може бути корисно в деяких випадках для запобігання постійному затриманню потоку, якщо семафор ніколи не встановлюється через стан помилки. Час очікування задається в мілісекундах. Якщо потрібно, щоб потік очікував весь час і семафор був встановлений, а час не закінчувався після певної затримки, потрібно використовувати WICED_WAIT_FOREVER для тайм-ауту.

Функції семафору доступні в документації в розділі “Components→ RTOS→Semaphores”.

Ви завжди повинні ініціалізувати семафор перед початком будь-яких потоків, які ним користуються. В іншому разі ви можете побачити непередбачувану поведінку.

5. М’ютекси

М’ютекс – це замок на певному ресурсі. Якщо ви запитаєте м’ютекс на ресурс, який вже заблокований іншим потоком, то ваш потік буде переходити до сну до виходу блокування. У задачах цієї роботи потрібно буде створите два різні потоки, які блимають одним світлодіодом. Без м’ютекса ви побачите дивну поведінку. За допомогою м’ютекса кожен з потоків отримує ексклюзивний доступ до світлодіодів.

Функції м’ютекса доступні в документації в розділі “Components→RTOS→Mutex”

Ви завжди повинні ініціалізувати м’ютекс, перш ніж запускати будь-які потоки, які його використовують. В іншому разі ви можете побачити непередбачувану поведінку.

Зауважте, що м’ютекс може бути розблокований лише тим самим потоком, який його заблокував.

6. Черги

Черга – це безпечний для потоків механізм передачі даних в інший потік. Черга – це FIFO: ви читаєте спереду і записуєте в хвіст. Якщо ви спробуєте прочитати порожню чергу, ваш потік буде призупинений, поки щось не буде записано в неї. Корисні дані в черзі (розмір кожного запису) та розмір черги (кількість записів) налаштовуються користувачем під час створення черги.

Для wiced_rtos_push_to_queue() потрібен параметр тайм-ауту. Це вступає в гру, якщо черга заповнена, коли ви намагаєтесь натиснути на неї. Час очікування дозволяє потоку продовжуватися через певний час, навіть якщо черга залишається повною. Це може бути корисно в деяких випадках, щоб запобігти постійній зупинці потоку, якщо черга залишається повною через стан помилки. Час очікування задається в мілісекундах. Якщо ви хочете, щоб потік нескінченно чекав місця в черзі, а не вичерпувався після певної затримки, використовуйте WICED_WAIT_FOREVER для очікування. Якщо ви хочете, щоб потік продовжувався негайно, якщо в черзі немає місця, тоді використовуйте WICED_NO_WAIT. Зауважте, що якщо функція відключається, значення не додається до черги.

Так само функція wiced_rtos_pop_from_queue() вимагає параметр тайм-ауту, щоб вказати, скільки часу потік повинен чекати, якщо черга порожня. Якщо ви хочете, щоб потік нескінченно чекав значення в черзі, а не продовжував виконання після певної затримки, використовуйте WICED_WAIT_FOREVER. Якщо ви хочете, щоб проєкт продовжувався негайно, якщо у черзі нічого немає, використовуйте WICED_NO_WAIT.

Також є функції перевірити, чи черга повна чи порожня, та визначити кількість записів у черзі.

Функції черги доступні в документації в розділі “Components→RTOS→Queues”.

Ви завжди повинні ініціалізувати чергу, перш ніж запускати будь-які потоки, які її використовують. В іншому разі ви можете побачити непередбачувану поведінку.

7. Таймери

Таймер RTOS дозволяє планувати функцію, яку потрібно запустити на визначений інтервал – наприклад, надсилайте свої дані в хмару кожні 10 секунд.

Під час налаштування таймера вказують функцію, яку потрібно запустити, і як часто її потрібно запускати. Функція, яку викликає таймер, приймає єдиний аргумент void* arg. Якщо для функції не потрібні аргументи, можна вказати NULL у функції ініціалізації таймера, але сама функція все одно повинна мати аргумент void* arg у своєму визначенні.

Потрібно зауважити, що функція виконується одноразово щоразу, коли таймер закінчується, а не постійно виконуючий потік, тому функція НЕ повинна мати цикл while(1) – він повинен просто запускатися та виходити кожного разу, коли таймер викликає його.

Таймер – це функція, а не потік. Тому потрібно переконатись, що не виходиться з основного потоку програми, якщо у проєкту немає інших активних потоків.

Функції таймера доступні в документації в розділі “Components→RTOS→RTOS Timers”.

8. Порядок роботи

1.Ознайомитись з документацією де описано функції RTOS.

2.Запустити проєкт мигання світлодіодом "01_thread". Замінити період затримки між миганням, ознайомитись з використаними функціями.

3.Запустити проєкт мигання світлодіодом після натиску кнопки "02_semaphore". Замінити світлодіод котрий моргає, ознайомитись з використаними функціями.

4.Запустити проєкт "03_mutex". Закоментувати USE_MUTEX та знову запустити проєкт, оцінити зміну поведінки світлодіода. Ознайомитись з використаними функціями.

5.Запустити проєкт "04_queue", натиснути кнопку на відлагоджувальній платі кілька разів щоб спостерігати, як відпрацьовує черга миганням світлодіоду. Ознайомитись з використаними функціями.

6.Запустити проєкт "05_timer". Змінити час перемикання світлодіодів у результаті спрацьовування таймера. Ознайомитись з використаними функціями.

9. Зміст звіту

1.Написати назву та мету виконання лабораторної роботи.

2.Описати хід виконання лабораторної роботи.

3.Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.

4.Написати висновок після виконання даної лабораторної роботи. Вказати основні моменти.

10. Контрольні питання

1. Чи потрібна затримка wiced_rtos_delay_milliseconds() у світлодіодному потоці? Чому так або чому ні?
2. Що станеться, якщо ви використовуєте значення 100 для тайм-ауту семафору? Чому?
3. Що станеться, якщо ви забудете розблокувати м’ютекс в одному з потоків? Чому?
4. Що станеться, якщо не зняти цикл while(1) з функції, що миготить світлодіодом? Чому?

5. Що станеться, якщо application_start не має циклу while(1)? Чому?
6. Чи потрібен цикл while (1) у application_start? Чому так або чому ні?

Використані джерела

1. Середовище – https://www.cypress.com/products/wiced-software.
2. Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.
3. Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.
4. Відлагоджувальна плата – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

Лабораторна робота № 4

ВИКОРИСТАННЯ БІБЛІОТЕК У СЕРЕДОВИЩІ WICED (JSON, DISPLAY)

Мета роботи: отримати розуміння про вміст бібліотеки WICED SDK. Крім того, зрозуміти формат файлу файлів Java Script Object Notation (який широко використовується в Інтернеті).

1. Бібліотеки WICED-SDK

Сьогодні життя занадто коротке, щоб розробити всі "речі", які ви можете включити у свій проєкт IoT. Щоб пришвидшити цикл розробки, WICED-SDK включає в себе купу коду для обробки багатьох завдань, які ви, можливо, захочете використовувати у своєму проєкті. Якщо ви заглянете в папку "бібліотеки" в робочій області SDK, ви знайдете такі підпапки:

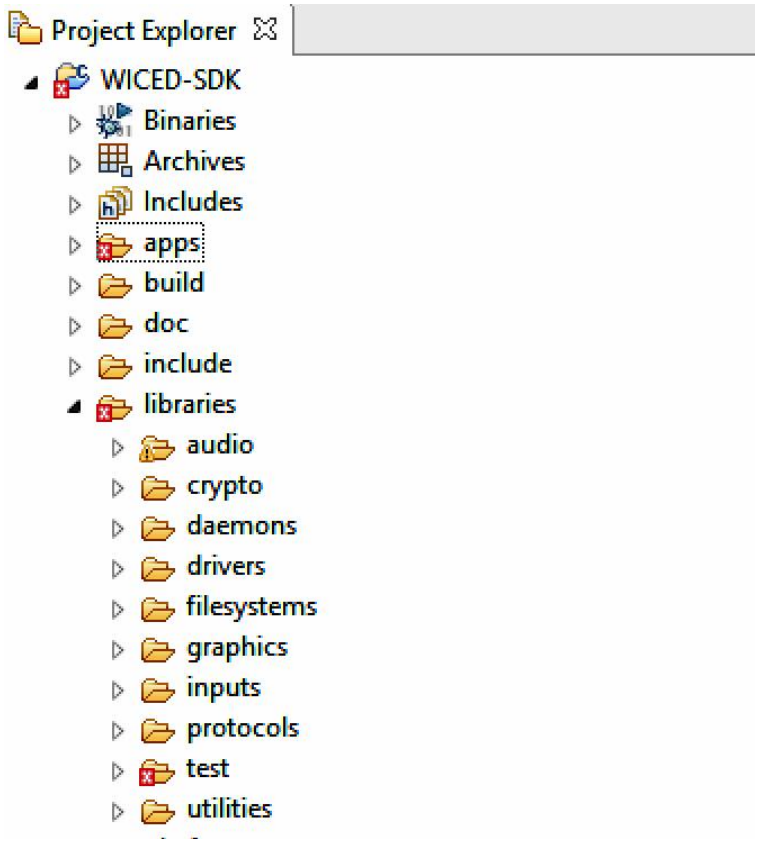


Рис. 1. Перелік об'єктів які містяться в папці "libraries"

- **audio**: містить підтримку Apollo (стандарт потокового аудіо) та кодеків, включаючи Free Lossless Audio Codec;
- **crypto**: утиліти криптографії: ECDH (Elliptic Curve Diffe-Hellman) та ECDSA (Алгоритм цифрового підпису Elliptic Curve);
- **daemons**: містить типові демони "Unix" для забезпечення підтримки мережі, включаючи HTTP-сервер, Gedday, TFTP, DHCP, DNS тощо;

- **drivers**: Містить файли апаратної підтримки для SPI флеш, USB тощо;
- **filesystems**: FAT, FILE та інші файлові системи, які можна записати на SPI флеш;
- **graphics**: підтримка дисплеїв OLED U8G;.
- **inputs**: драйвери для кнопок та GPIO;
- **protocols**: підтримка протоколів рівня додатків, включаючи HTTP, COAP, MQTT і т. д.;
- **test**: Інструменти для перевірки продуктивності мережі, iPerf, malloc, TraceX, аудіо;
- **utilities**: підтримка JSON, пов'язаних списків, консолі, printf, буферів тощо.

2. U8G Графіка

В роботі використовуватимемо графічну бібліотеку для відображення інформації на OLED-дисплеї, котрий є на макетній платі. Дисплей – це OLED-екран із 128×64 пікселями та чіп драйвера Freetronics SSD1306 на базі I2C. Інтерфейс I2C Freetronics SSD1306 підключений до тієї ж шини I2C, що і PSoC 4, але за іншою адресою I2C. Freetronics SSD1306 на нашому дисплеї має фіксовану I2C адресу 0×3C.

OLED дозволяє малювати фігури та писати текст. Якщо ви використовуєте таку функцію, як u8g_DrawLine (), координати *x* і *y* починаються у верхньому лівому куті (0, 0).

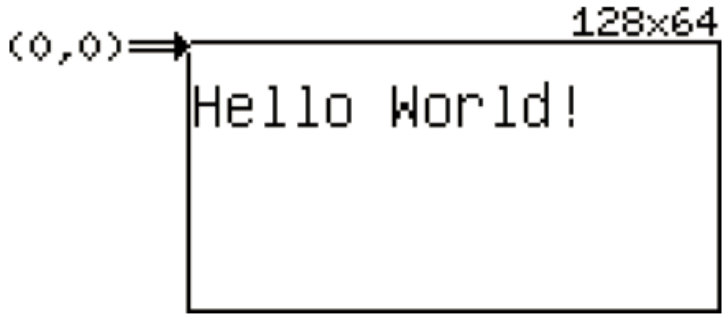


Рис. 2. Приклад зображення на дисплеї

Оновлення завжди відбуваються в "цикл зображення", де функції малювання повторно виконуються до належного оновлення OLED. Дві функції бібліотеки – u8g_FirstPage () та u8g_NextPage () – роблять це дуже просто, використовуючи цикл do-while, як ви побачите нижче. Це робиться для того, щоб дані, необхідні для заповнення всього екрану (128×64 = 8192 пікселів), могли бути розбиті на кілька "сторінок", що зменшує об'єм пам'яті, необхідний хост-процесору для створення та надсилання зображення.

Порядок малювання тексту на дисплеї:

1. Налаштування структури типу u8g_t. Вказівник на цю структуру буде першим аргументом майже у всіх викликах функції u8g, використовується.

2. Налаштуйте та ініціалізуйте структуру I2C для OLED-дисплея. Для нашого обладнання:

a) I2C port = WICED_I2C_2;
b) I2C address = 0×3C;
c) I2C address with = I2C_ADDRESS_WIDTH_7BIT;
d) Flags = 0;

e) Speed mode = I2C_STANDARD_SPEED_MODE.

3. Ініціалізуйте пристрій I2C за u8g_init_wiced_i2c_device. Ця функція приймає покажчик на структуру I2C з кроку 2.

4. Ініціалізуйте функції зв'язку, викликаючи u8g_InitComFn. Функція отримує покажчик на структуру u8g, створену на кроці 1, вказівник на структуру u8g_dev_t, яка вказує тип дисплея, і вказівник функції зв'язку. Для нашого обладнання, якщо у вас є структура дисплея під назвою "display", виклик виглядає так:

u8g_InitComFn (& display, & u8g_dev_ssd1306_128x64_i2c, u8g_com_hw_i2c_fn).

5. Виберіть шрифт за допомогою u8g_SetFont. Він отримує вказівник на структуру u8g та назву шрифту.

Усі шрифти перераховані у файлі u8g_font_data.c у каталозі бібліотеки графіки. У прикладах використовується u8g_font_unifont, але сміливо експериментуйте з іншими, якщо хочете.

6. Встановіть позицію, використовуючи u8g_SetFontPosTop, u8g_SetFontPosBottom або u8g_SetFontPosCenter.

Ці функції визначають, де малюються символи відносно початкових координат, визначених у функції DrawStr, описаній нижче. u8g_SetFontPosTop означає, що верхня частина першого символу буде вказаною координатою.

7. Кожен раз, коли ви бажаєте вивести рядок, виконайте вказані нижче дії:

a) виберіть сторінку для відображення рядка за допомогою u8g_FirstPage;

b) намалюйте рядок за допомогою u8g_DrawStr. Ви повинні викликати це неодноразово, доки u8g_NextPage не поверне 0. Функція u8g_DrawStr не вказує на структуру u8g, координату X, координату Y та рядок для друку;

c) щоб ефективно малювати на екрані, графічний пакет u8g ділить екран на кілька горизонтальних розділів або сторінок. Використання u8g_FirstPage та u8g_NextPage повторює ці сторінки для малювання повного зображення.

Як приклад, якщо припустити структуру відображення під назвою "display" та структуру I2C під назвою "display_i2c", далі буде надруковано рядок "Cypress":

```
u8g_init_wiced_i2c_device(&display_i2c);
u8g_InitComFn(&display, &u8g_dev_ssd1306_128x64_i2c, u8g_com_hw_i2c_fn);
u8g_SetFont(&display, u8g_font_unifont);
u8g_SetFontPosTop(&display);
u8g_FirstPage(&display);
do
{
u8g_DrawStr(&display, 0, 10, "Cypress");
} while (u8g_NextPage(&display));
```

Крім того, ви повинні включити "u8g_arm.h" у .c файлі, і ви повинні включити бібліотеку u8g у файлі .mk, щоб мати доступ до функцій бібліотеки:

У <project> .c: #include u8g_arm.h

У <project> .mk: $ (NAME) _COMPONENTS: = graphics/u8g.

Примітка: u8g_arm.h включає wiced.h, тому вам не потрібно включати wiced.h окремо, але не завадить включати обидва.

3. JavaScript Object Notation (JSON)

JSON – це формат відкритого стандарту, який використовує читанийлюдиною текст для передачі даних. Це фактичний стандарт передачі даних у хмару/з неї. JSON підтримує такі типи даних:

- плаваюча точка з подвійною точністю;
- стрічки;
- булева (true чи false);
- масиви (використовуйте "[]" для визначення масиву зі значеннями, розділеними ",").
- ключ/значення пари як "ключ": значення (використовуйте "{}", щоб описати пари) на "," розділяючи пари.

Значення пари ключ/значення можуть бути масивами або можуть бути іншими парами ключ/значення.

Масиви можуть містити пари ключ/значення.

Наприклад, коректний файл JSON виглядає так:

```
{
  "name" : "alan",
  "age" : 49,
  "badass" : true,
  "children": ["Anna","Nicholas"],
  "address" : {
        "number":201,
        "street": "East Main Street",
        "city": "Lexington",
        "state":"Kentucky",
        "zipcode":40507
  }
}
```

Зауважте, що повернення каретки та пробіли (крім самих рядків) не мають значення. Наприклад, вищевказаний код JSON може бути записаний у вигляді:

```
{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],"address":{"number":201,"street":"East Main Street","city":"Lexington","state":"Kentucky","zipcode":40507}} .
```

Хоча людині це важче прочитати, легше створити таку стрічку в прошивці, коли потрібно надсилати документи JSON.

На жаль, лапки означають щось для компілятора C, тому якщо ви включаєте рядок JSON всередині програми C, вам потрібно уникнути цитат, що знаходяться всередині JSON, зі зворотним нахилом (\). Вищенаведений JSON буде представлений так всередині програми C:

```
{\"name\":\"alan\",\"age\":49,\"badass\":true,\"children\":[\"Anna\",\"Nicholas\"],\"address\":{\"number\":201,\"street\":\"East Main Street\",\"city\":\"Lexington\",\"state\":\"Kentucky\",\"zipcode\":40507}} .
```

Є вебсайт, який можна використовувати для перевірки помилок JSON. Його можна знайти за адресою: https://jsonformatter.curiousconcept.com.

4. Створення JSON

Якщо вам потрібно створити JSON для надсилання (наприклад, у хмару), ви можете створити рядок за допомогою snprintf. Ви можете використовувати стандартне форматування printf для заміни значень змінних. Наприклад, для передачі температури у вигляді величини з плаваючою точкою від пристрою IoT до постачальника хмарних послуг може використовуватися наступне:

char json[100];

snprintf(json, sizeof(json), "{\"state\" : {\"reported\" : {\"temperature\":%.1f} } }", psoc_data.temperature).

% .1f замінюється в рядку на psoc_data.temperature як значення з плаваючою комою з одним місцем після десяткової. Якщо фактична температура становить 25,4, то результуюча рядок, створена в масиві json, буде:

{"state" : {"reported" : {"temperature":25.4} } }

5. Читання JSON (WICED-JSON Parsers)

Якщо вам потрібно отримати JSON (наприклад, із хмари), а потім витягнути певне значення, ви можете використовувати аналізатор JSON. Аналізатор прочитає JSON і знайде та поверне значення для вказаних вами ключів.

У бібліотеці WICED є два парсери JSON: cJSON та JSON_parser. cJSON – це аналізатор моделей об'єктів документа, тобто він читає весь JSON залпом. JSON_parser є ітераційним і, отже, дає змогу аналізувати більші файли. Ви можете знайти їх у SDK під libraries/utilities/. cJSON простіший у використанні, але може знадобитися використання JSON_parser для дуже великих файлів JSON. Для пристроїв IoT cJSON майже завжди буде достатнім.

6. Бібліотека WICED-cJSON

cJSON читає та обробляє весь документ за один раз, а потім дозволяє отримувати доступ до даних у документі за допомогою API, щоб знайти елементи в JSON. Ви можете подивитися файл README, який знаходиться в libraries/utilities/cJSON/README. Ви пройдете ієрархію JSON один рівень за часом, поки не досягнете ключової пари значень, яка вас цікавить. Функції, як правило, повертають вказівник на структуру типу cJSON, яка містить елементи для кожного типу даних повернення (тобто valuestring, valueint, valuedouble, тощо).

Наприклад, якщо у вас є масив char, що називається data з JSON, пов'язаними з Alan:

{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],"address":{"number":201,"street":"East Main Street","city":"Lexington","state":"Kentucky","zipcode":40507}} .

Код для отримання поштового індексу Алана виглядатиме приблизно так:

```
#include <wiced.h>
#include <cJSON.h>
#include <stdint.h>
void application_start()
{
        int zipcodeValue;
        cJSON *root = cJSON_Parse(data); //Read the JSON
        cJSON *address = cJSON_GetObjectItem(root,"address"); // Search for the key "address"
```

```
    cJSON *zipcode = cJSON_GetObjectItem(address,"zipcode"); // Search for the key
    "zipcode" under    //address
    zipcodeValue = (float) zipcode->valueint; // Get the integer value associated with the key
    zipcode
}
```

Щоб включити бібліотеку cJSON у свій проєкт:

1. Включіть cJSON.h у вихідний файл .c:

```
#include <cJSON.h> .
```

2. Додайте його до Makefile:

```
$(NAME)_COMPONENTS := utilities/cJSON .
```

7. Порядок роботи

1. Ознайомитись папкою бібліотеки, щоб побачити, які функції доступні.
2. Переглянути документацію графічної бібліотеки.
3. Запустити проєкт виведення стрічки на екран "ww101/04/02_hello". Змінити стрічку, змінити шрифти, ознайомитись з використаними функціями.
4. Запустити проєкт "ww101/04/03_cjson" для аналізу документу JSON за допомогою бібліотеки "cJSON" та виводу значень у термінал. Змінити дані, ознайомитись з використаними функціями.

8. Зміст звіту

1. Написати назву та мету виконання лабораторної роботи.
2. Описати хід виконання лабораторної роботи.
3. Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.
4. Написати висновок після виконання даної лабораторної роботи. Вказати основні моменти.

9. Контрольні питання

1. Що міститься в папці бібліотеки?
2. В якій послідовності виводиться інформація на дисплей?
3. Що таке JSON?
4. Для чого використовується JSON?

10. Використані джерела

1. Середовище – https://www.cypress.com/products/wiced-software.
2. Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.
3. Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.
4. Відлагоджувальна плата – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

Лабораторна робота № 5
ПІД'ЄДНАННЯ ДО ТОЧКИ ДОСТУПУ WI-FI

Мета роботи: зрозуміти основи роботи Wi-Fi (STA) та підключення до точки доступу Wi-Fi (AP). Почати ознайомлення з мережевим стеком TCP/IP, і отримати базове розуміння перших трьох шарів еталонної моделі взаємодії відкритих систем OSI. Отримати основне розуміння шару Wi-Fi, який обробляє з'єднання та шифрування. А також зрозуміти деякі основи IP-мережі (адреси, мережеві маски) та роль таблиці WICED Device Configuration (DCT).

1. Мережевий стек TCP/IP

Практично всі складні системи управляють загальною складністю, поділяючи систему на шари. "Мережевий стек" або, точніше, "Мережевий стек TCP/IP" – це саме те, що ієрархічна система для надійної комунікації через безліч мережевих середовищ (Wi-Fi, Ethernet тощо). Кожен шар ізолює користувача цього шару від складності шару під ним і спрощує зв'язок для шару над ним. Ви можете почути про мережеву модель OSI – ще один подібний спосіб опису мережевих шарів; однак, простіше уявити IP-мережі за допомогою моделі TCP/IP.

Кожен шар приймає вхід шару над ним, а потім вбудовує цю інформацію в один або кілька блоків даних протоколу (PDU) цього рівня. PDU – елементарна одиниця даних для цього шару: наприклад, шар Datalink приймає пакет IP і ділить його на 1 або більше кадрів рівня шарів зв'язку Wi-Fi. Фізичний шар займає шари кадрів Datalink Layer і ділить їх на біти.

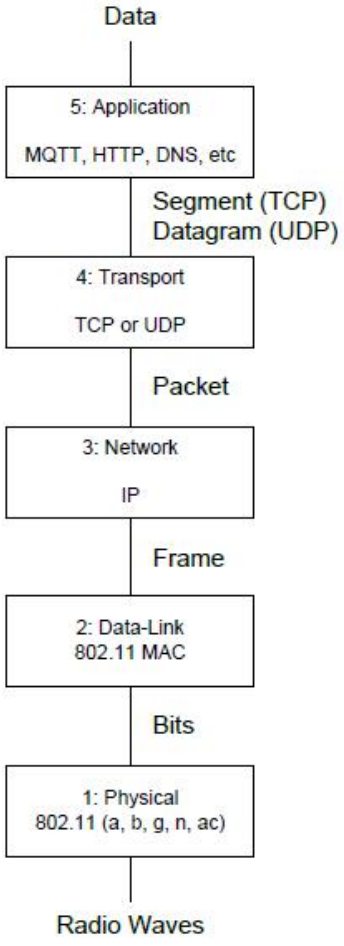


Рис 1. Потік даних в OSI системі

3. Основи Wi-Fi

Є два вузли мережі Wi-Fi: станція (тобто пристрій IoT) і точка доступу (тобто бездротовий маршрутизатор). Для того, щоб станція підключилася до точки доступу Wi-Fi, вона повинна знати таку інформацію: **SSID**, **схема шифрування** та **пароль** (якщо потрібно). Чіп WICED займається вибором правильної смуги та каналу. Щоб надіслати дані, всі кадри даних Wi-Fi Datalink позначені мітками з MAC-адресою джерела та MAC-адресою призначення.

SSID (назва бездротової мережі) – SSID означає Ідентифікатор набору послуг. SSID – це мережеве ім'я, яке складається з 1–32 байт. Ім'я не повинне бути читабельним для людини (наприклад, ASCII), але оскільки воно є некодованими байтами, воно є ефективно чутливим до регістру (будьте обережні).

Діапазон (або 2,4 ГГц, або 5 ГГц) – радіостанції Wi-Fi кодують значення 1 і 0 з однією з кількох різних схем модуляції залежно від типу мережі Wi-Fi (a, b, g, n, ac, ax) та режиму роботи. Типи кодування прозорі для вашої програми IoT, оскільки мікросхема, радіо- та мікропрограмне забезпечення віртуалізують це для вас. Далі дані передаються в діапазон 2,4 ГГц або 5 ГГц (який діапазон важливий). Зауважте, що діапазон 5 ГГц має більшу пропускну здатність і меншу затримку, але менший радіус дії, тоді як для діапазону 2,4 ГГц навпаки.

Номер каналу – доступні канали мають діапазон (2,4 ГГц проти 5 ГГц) та географічне розташування (місцеположення). Крім того, FCC регулює, які канали та смуги можуть використовуватися для різних діючих регіонів світу. На шарі Wi-Fi це налаштовано за допомогою параметра коду країни, який відображає набір доступних каналів для цього регіону. 2,4 ГГц досить простий, є канали 1–14 з 1–11 доступними по всьому світу. 5 ГГц – це конкретні регіональні органи, і регуляторні органи (наприклад, FCC) доручать, які канали ви можете використовувати, залежно від регіону. Однак з погляду станції нічого з цього не має значення, оскільки при спробі приєднатись до SSID WICED-SDK сканує всі канали, шукаючи правильний SSID.

Шифрування (Open, WEP, WPA, WPA2) – для забезпечення безпеки мереж Wi-Fi зазвичай використовується шифрування рівня передачі даних. Типи мережевого шифрування – це **Open** (тобто відсутність безпеки), **Wired Equivalent Privacy (WEP)**, яка не є повністю захищеною (але може бути нормальною для деяких типів обмежених застарілих додатків), **Wi-Fi Protected Access (WPA)** та **WPA2**, яка має значною мірою переміщена з WPA. З цього моменту ми будемо називати шифрування WPA, але ми маємо на увазі WPA2. Існує дві версії WPA: одна називається "Personal" або "Pre Shared Key" (PSK) і одна називається "Enterprise".

WEP і WPA PSK використовують пароль, який називають ключем, для шифрування даних. Схема шифрування WEP не рекомендується, оскільки її дуже легко зкомпрометувати (наприклад, використовувати такі інструменти, як Wireshark та AirSnort). Схема ключа PSK WPA дуже безпечна, оскільки вона використовує AES (Advanced Encryption Standard). Однак обмін ключами є болючим, незахищеним процесом, оскільки це означає, що всі мають один ключ. Для вирішення ключової проблеми розподілу більшість корпоративних мережевих рішень використовують WPA Enterprise, який вимагає використання сервера RADIUS для обробки автентифікації кожної станції окремо.

Media Access Control (MAC) Address – MAC-адреса Wi-Fi – це 48-розрядний унікальний номер, що складається з OUI (організаційно унікального ідентифікатора) та ідентифікатора станції. Перші три байти MAC-адреси – це поле OUI, яке IEEE призначає унікальним для виробника (наприклад, Cypress). Для того, щоб шар даталінку надсилав кадр, він повинен адресувати кадр з MAC-адресою джерела та призначення. Інші пристрої в мережі передаватимуть кадри лише у вищі рівні стеку, які адресовані їм. Пам’ятайте, що шар Datalink нічого не знає про вищі шари (наприклад, IP). Нарешті, найзначніший біт найзначнішого байта (наприклад, біт 47) визначає адресу багатоадресної передачі (Group), а спеціальна адреса всіх 1-х (наприклад, ff: ff: ff: ff: ff: ff) – це адреса широкомовної передачі (надіслати до кожного).

Рівень datalink повинен мати можливість визначити MAC-адресу певної IP-адреси, щоб надсилати дані на цю IP-адресу в мережу Wi-Fi. Для з’ясування цього відображення існує протокол під назвою Address Resolution Protocol (ARP).

ARP – всередині кожного пристрою є таблиця ARP, яка має карту MAC-адреси до IP-адреси. Для виявлення MAC-адреси IP-адреси в мережу транслюється "ARP-запит". Усі пристрої, підключені до мережі, слухають запити ARP. Якщо ви чуєте запит ARP з вказаною вами IP-адресою, ви відповідаєте своєю MAC-адресою. З цього моменту обидві сторони додають цю інформацію до своєї таблиці ARP (а якщо ви чуєте інші ARPing, ви можете також оновити свою таблицю). Ідея цієї схеми полягає в тому, що якщо ви ARP для IP-адреси, яка відсутня у вашій локальній мережі, маршрутизатор відповість своєю MAC-адресою.

4. IP-мережі

Інтернет – це сітка взаємопов'язаних мереж IP. Хмара – це весь Інтернет, який доступний вашій мережі, але може означати і сервери, які приєднані до мережі десь в Інтернеті.

Усі пристрої в Інтернеті мають реальну IP-адресу і належать до (IP) мережі, визначеної мережевою маскою. Маршрутизатори – це пристрої, що з'єднують IP-мережі, беручи IP-пакети з однієї мережі та пересилаючи їх до правильної наступної мережі. Це складне завдання, яке виходить за межі цієї лабораторної роботи.

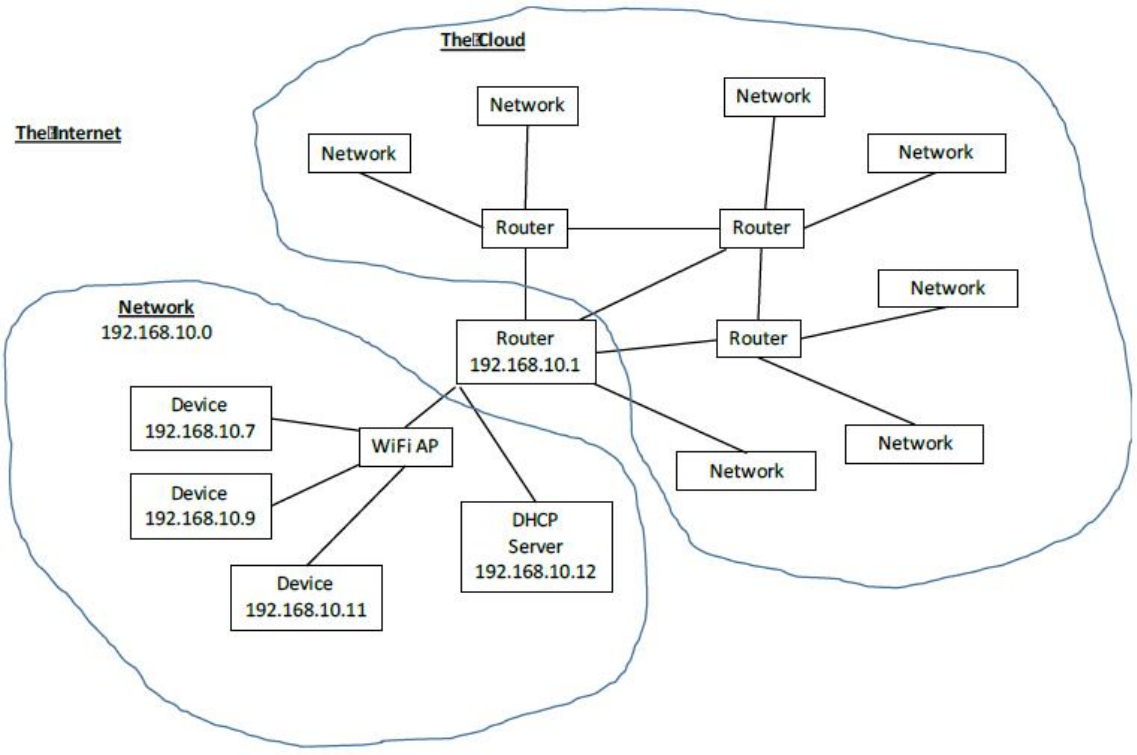


Рис. 2. Візуалізація організації мережі Інтернет

IP-адреса однозначно ідентифікує окремий пристрій з 32-бітовим числом, яке, як правило, виражається чотирма шістнадцятибайтовими розділеними періодами. Наприклад 192.168.15.7. IP-адреси поділяються на дві частини: мережева адреса (яка є першим *x* числом біт) та адреса клієнта, які є останніми 32-*x* бітами. Маска мережі визначає розкол мережі/клієнта. Наприклад мережна маска для 192.168.15. * дорівнює 255.255.255.0.

Мережа IP (іноді її називають IP-підмережа) – це сукупність пристроїв, які мають спільну мережеву адресу, наприклад усі пристрої на 192.168.15. * (мережна маска 255.255.255.0) є частиною однієї і тієї ж мережі IP.

Найчастіше IP-адреси пристроїв типу IoT динамічно призначаються сервером динамічного протоколу управління хостом (DHCP). Щоб динамічно призначити DHCP-адресу, спершу ви надсилаєте дейтаграму рівня 2 рівня із запитом IP-адреси (DHREQUEST). Коли сервер DHCP чує запит, він відповідає на необхідну інформацію. DHCP інтегрований у WICED та обробляє цей обмін інформацією для вас автоматично, коли він увімкнений.

5. Таблиця конфігурації пристрою (DCT)

Таблиця конфігурації пристрою – це розділ WICED-пам'яті із заздалегідь визначеним форматом, який використовується для зберігання основної інформації про систему (тобто клієнтський AP APID, клієнтський пароль AP та ін.). Він також може використовуватися для зберігання інформації про вашу програму. DCT використовується вбудованим програмним забезпеченням WICED, щоб "зробити правильно". Наприклад, wiced_network_up () зчитує мережеву інформацію з DCT і підключається до вказаної мережі.

Таблиця будується в процесі компіляції та записується у пам'ять разом із вашим додатком. DCT також може бути модифікований (і записаний) під час роботи вашої програми.

Створюючи WICED-додаток, ви можете використовувати DCT за замовчуванням або створити спеціальний розділ. Щоб заздалегідь налаштувати розділ Wi-Fi таблиці DCT, вам потрібно створити .h-файл (зазвичай називається wi-fi_config_dct.h) з правильними #defines. Потім потрібно додати наступний рядок до makefile, щоб ваш власний DCT був включений у збірку:

WI-FI_CONFIG_DCT_H := wi-fi_config_dct.h.

Ви можете отримати шаблон для файлу в каталозі "include/default_wi-fi_config_dct.h". Зауважте, що ім'я файлу в більшості проєктів жорстко закодовано як "wi-fi_config_dct.h", тому його потрібно перейменувати.

```
#pragma once

/* This is the soft AP used for device configuration */
#define CONFIG_AP_SSID       "WICED_AWS"
#define CONFIG_AP_CHANNEL    1
#define CONFIG_AP_SECURITY   WICED_SECURITY_WPA2_AES_PSK
#define CONFIG_AP_PASSPHRASE "12345678"

/* This is the soft AP available for normal operation (if used)*/
#define SOFT_AP_SSID         "WICED Device"
#define SOFT_AP_CHANNEL      7
#define SOFT_AP_SECURITY     WICED_SECURITY_WPA2_AES_PSK
#define SOFT_AP_PASSPHRASE   "WICED_PASSPHRASE"

/* This is the default AP the device will connect to (as a client)*/

#define CLIENT_AP_SSID       "Guest"
#define CLIENT_AP_PASSPHRASE ""

#define CLIENT_AP_BSS_TYPE   WICED_BSS_TYPE_INFRASTRUCTURE
#define CLIENT_AP_SECURITY   WICED_SECURITY_OPEN
#define CLIENT_AP_CHANNEL    6
#define CLIENT_AP_BAND       WICED_802_11_BAND_2_4GHZ

/* This is the network interface the device will work with */
#define WICED_NETWORK_INTERFACE   WICED_STA_INTERFACE
```

Пристрій може працювати в трьох режимах, як показано у таблиці вище: Конфігурація AP (рядки 4–7), Normal AP (10-13) та Client Mode (тобто STA) (рядки 17–23). Також можливо мати декілька мережевих інтерфейсів та підтримувати Wi-Fi та Ethernet (рядок 26). AP-конфігурація використовується для пристроїв, які хочуть дозволити іншим пристроям підключатися до них для здійснення конфігурації системи WICED через Wi-Fi. Номінальна точка доступу використовується для пристроїв, які будуть функціонувати в якості точки доступу Wi-Fi під час нормальної роботи. Клієнт використовується для пристроїв, які підключаться до існуючої мережі Wi-Fi як станція. Для цілей цієї лабораторної ми будемо лише КЛІЄНТОМ, тому вам потрібно буде лише змінити 17–23.

Щоб знайти визначення (або можливі визначення) #defines, можна виділити, клацнути правою кнопкою миші та вибрати "Open Declaration". Наприклад, якщо ви відкриєте декларацію "WICED_SECURITY_OPEN", вам потрібно буде:

```
144 typedef enum
145 {
146     WICED_SECURITY_OPEN           = 0,
147     WICED_SECURITY_WEP_PSK        = WEP_ENABLED,
148     WICED_SECURITY_WEP_SHARED     = ( WEP_ENABLED   | SHARED_ENABLED ),
149     WICED_SECURITY_WPA_TKIP_PSK   = ( WPA_SECURITY  | TKIP_ENABLED ),
150     WICED_SECURITY_WPA_AES_PSK    = ( WPA_SECURITY  | AES_ENABLED ),
151     WICED_SECURITY_WPA_MIXED_PSK  = ( WPA_SECURITY  | AES_ENABLED | TKIP_ENABLED ),
152     WICED_SECURITY_WPA2_AES_PSK   = ( WPA2_SECURITY | AES_ENABLED ),
153     WICED_SECURITY_WPA2_TKIP_PSK  = ( WPA2_SECURITY | TKIP_ENABLED ),
154     WICED_SECURITY_WPA2_MIXED_PSK = ( WPA2_SECURITY | AES_ENABLED | TKIP_ENABLED ),
155
156     WICED_SECURITY_WPA_TKIP_ENT   = ( ENTERPRISE_ENABLED | WPA_SECURITY  | TKIP_ENABLED ),
157     WICED_SECURITY_WPA_AES_ENT    = ( ENTERPRISE_ENABLED | WPA_SECURITY  | AES_ENABLED ),
158     WICED_SECURITY_WPA_MIXED_ENT  = ( ENTERPRISE_ENABLED | WPA_SECURITY  | AES_ENABLED | TKIP_ENABLED ),
159     WICED_SECURITY_WPA2_TKIP_ENT  = ( ENTERPRISE_ENABLED | WPA2_SECURITY | TKIP_ENABLED ),
160     WICED_SECURITY_WPA2_AES_ENT   = ( ENTERPRISE_ENABLED | WPA2_SECURITY | AES_ENABLED ),
161     WICED_SECURITY_WPA2_MIXED_ENT = ( ENTERPRISE_ENABLED | WPA2_SECURITY | AES_ENABLED | TKIP_ENABLED ),
162
163     WICED_SECURITY_IBSS_OPEN      = ( IBSS_ENABLED ),
164     WICED_SECURITY_WPS_OPEN       = ( WPS_ENABLED ),
165     WICED_SECURITY_WPS_SECURE     = ( WPS_ENABLED | AES_ENABLED),
166
167     WICED_SECURITY_UNKNOWN        = -1,
168
169     WICED_SECURITY_FORCE_32_BIT   = 0x7fffffff
170 } wiced_security_t;
```

Інформація про DCT відображається у пам’яті WICED-SDK. Як правило, вам не потрібно буде знати про це, оскільки ви просто вибираєте свої налаштування у файлі wiced_config_dct.h, але якщо ви хочете прочитати/змінити деякі налаштування DCT з прошивки, вам потрібно буде зрозуміти, як зберігаються значення у пам’яті.

WICED-SDK забезпечує заздалегідь задану структуру для відображення DCT у файлі platform_dct.h, який можна знайти в папці WICED/platform/include).

```
740 typedef struct
741 {
742     platform_dct_header_t              dct_header;
743     platform_dct_mfg_info_t            mfg_info;
744     platform_dct_security_t            security_credentials;
745     platform_dct_wifi_config_t         wifi_config;
746     platform_dct_ethernet_config_t     ethernet_config;
747     platform_dct_network_config_t      network_config;
748     platform_dct_bt_config_t           bt_config;
749     platform_dct_p2p_config_t          p2p_config;
750     platform_dct_ota2_config_t         ota2_config;
751     platform_dct_version_t             dct_version;
752 } platform_dct_data_t;
753
```

Як видно з таблиці вище, DCT розділений на секції. Наприклад, wi-fi_config – це структура типу platform_dct_wi-fi_config_t, яка містить інформацію про конфігурацію Wi-Fi, включаючи відомі точки доступу. Якщо ви клацнете правою кнопкою миші та зробите Open Declaration на platform_dct_wi-fi_config_t, ви побачите:

```
611 typedef struct
612 {
613     wiced_bool_t              device_configured;
614     wiced_config_ap_entry_t   stored_ap_list[CONFIG_AP_LIST_SIZE];
615     wiced_config_soft_ap_t    soft_ap_settings;
616     wiced_config_soft_ap_t    config_ap_settings;
617     wiced_country_code_t      country_code;
618     wiced_aggregate_code_t    aggregate_code;
619     wiced_mac_t               mac_address;
620     uint8_t                   padding[2];  /* ensure 32bit aligned size */
621 } platform_dct_wifi_config_t;
```

Другий запис "storage_ap_list" – це масив типу "wiced_config_ap_entry_t". Перший елемент (тобто індекс 0) цього масиву містить інформацію для точки доступу, до якої STA підключається як клієнт. Якщо ви клацнете правою кнопкою миші на wiced_config_ap_entry_t і відкриєте декларацію, ви побачите:

```
571 typedef struct
572 {
573     wiced_ap_info_t details;
574     uint8_t         security_key_length;
575     char            security_key[ SECURITY_KEY_SIZE ];
576 } wiced_config_ap_entry_t;
```

Перший запис у цій структурі (деталі, що є структурою типу wiced_ap_info_t) містить дані про точку доступу, до якої підключиться клієнт. Якщо ви клацнете правою кнопкою миші на wiced_ap_info_t і зробите Open Declaration, ви побачите:

```
162 typedef struct wiced_ap_info
163 {
164     wiced_ssid_t             SSID;           /**< Service Set Identification (i.e. Name of Access Point)
165     wiced_mac_t              BSSID;          /**< Basic Service Set Identification (i.e. MAC address of Ac
166     int16_t                  signal_strength; /**< Receive Signal Strength Indication in dBm. <-90=Very poo
167     uint32_t                 max_data_rate;  /**< Maximum data rate in kilobits/s
168     wiced_bss_type_t         bss_type;       /**< Network type
169     wiced_security_t         security;       /**< Security type
170     uint8_t                  channel;        /**< Radio channel that the AP beacon was received on
171     wiced_802_11_band_t      band;           /**< Radio band
172     struct wiced_ap_info*    next;           /**< Pointer to the next scan result
173 } wiced_ap_info_t;
```

Багато записів у цій структурі також є структурами. Ви можете дослідити кожну з окремих структур, щоб побачити, які значення вони містять.

DCT може існувати у вигляді серії флеш-рядків всередині процесора додатка (тобто, якщо він має внутрішню пам’ять), або він може існувати в мікросхемі пам’яті, приєднаній до чипа Wi-Fi.

Для того, щоб прочитати з DCT, вам потрібно викликати функцію wiced_dct_read_lock (), яка прочитає DCT в буфер оперативної пам’яті, який ви зможете змінити, а потім записати на флеш за допомогою функції wiced_dct_write().

Ви надаєте виклик wiced_dct_read_lock() з вказівником на покажчик на порожню структуру, яка буде заповнена даними Wi-Fi DCT. Тип структури залежить від того, який

розділ DCT ви хочете прочитати (розділ є параметром функції wiced_dct_read_lock ()). Наприклад, якщо ви хочете прочитати DCT_WI-FI_CONFIG_SECTION, тоді тип вказівника буде platform_dct_wi-fi_config_t.

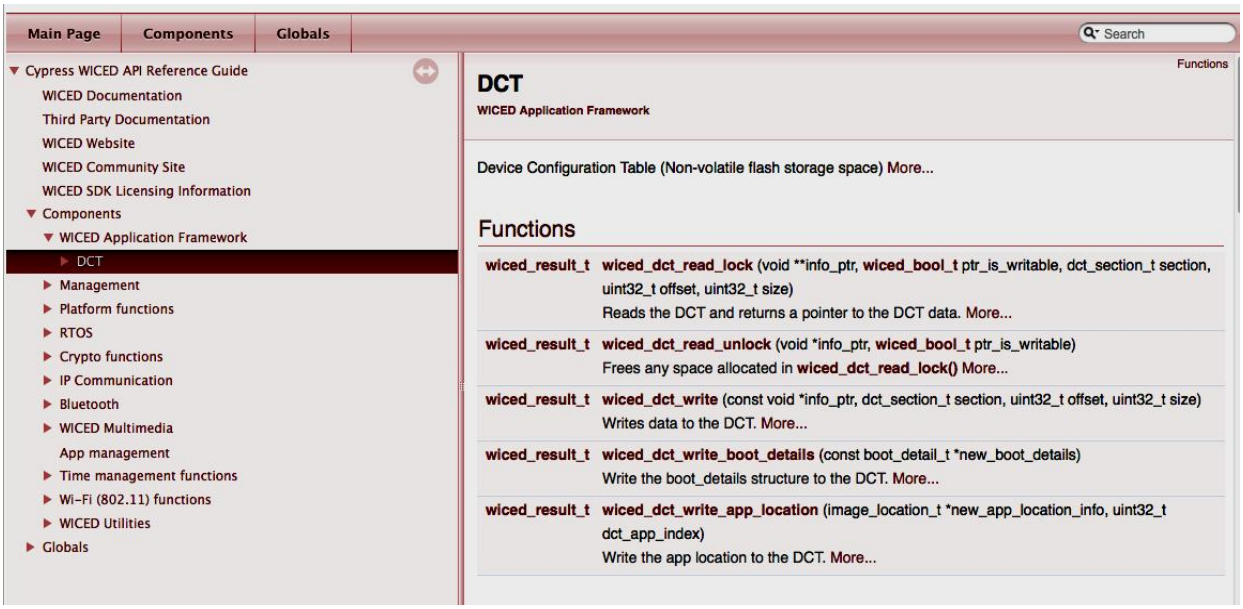
Ви можете знайти список назв розділів у файлі wiced_dct_common.h, який знаходиться у WICED/platform/MCU. Ось наявні розділи:

```
74 /* DCT section */
75 typedef enum
76 {
77     DCT_APP_SECTION,
78     DCT_SECURITY_SECTION,
79     DCT_MFG_INFO_SECTION,
80     DCT_WIFI_CONFIG_SECTION,
81     DCT_INTERNAL_SECTION, /* Do not use in apps */
82     DCT_ETHERNET_CONFIG_SECTION,
83     DCT_NETWORK_CONFIG_SECTION,
84     DCT_BT_CONFIG_SECTION,
85     DCT_P2P_CONFIG_SECTION,
86     DCT_OTA2_CONFIG_SECTION,
87     DCT_VERSION_SECTION, /* Do not use in apps */
88 } dct_section_t;
```

Коли ви закінчите з копією оперативної пам’яті DCT, вам потрібно звільнити її, викликавши функцію wiced_dct_read_unlock().

Якщо флеш "внутрішній" і безпосередньо доступний процесору, ви можете викликати wiced_dct_read_lock() з параметром запису, встановленим на false, і в цьому випадку wiced_dct_read_lock() дасть вам вказівник на флеш замість копіювання в оперативну пам’ять. Зробити це можна лише в тому випадку, якщо ви просто збираєтеся читати DCT. Тобто, якщо ви хочете мати можливість записувати в DCT, для параметра, що записується, потрібно встановити значення true.

Функції DCT задокументовані у розділі Components→WICED Application Framework→DCT.



6. WICED Wi-Fi SDK

Щоб підключитися до мережі Wi-Fi, потрібно викликати функцію wiced_network_up (). Цей виклик API має три параметри: мережевий інтерфейс, який потрібно використовувати; який метод використовувати для отримання вашої IP-адреси тощо; і які статичні IP-параметри використовувати (або NULL). Ось API з документації WICED:

```
wiced_result_t wiced_network_up ( wiced_interface_t          interface,
                                  wiced_network_config_t     config,
                                  const wiced_ip_setting_t * ip_settings
                                )
```

Параметр wiced_interface_t вказує, який мережевий інтерфейс використовувати. WICED-SDK підтримує можливість одночасно використовувати декілька мереж, наприклад, Wi-Fi та Ethernet. Щоб знайти визначення, перейдіть до визначення wiced_interface_t (виділивши, клацнувши правою кнопкою миші та вибравши Open Declaration) в SDK. Для цілей цього класу ми завжди будемо використовувати WICED_STA_INTERFACE, тобто ми завжди будемо станцією (тобто клієнтом), а не точкою доступу.

```
typedef enum
{
    WICED_STA_INTERFACE      = WWD_STA_INTERFACE,           /**< STA or Client Interface  */
    WICED_AP_INTERFACE       = WWD_AP_INTERFACE,            /**< softAP Interface         */
    WICED_P2P_INTERFACE      = WWD_P2P_INTERFACE,           /**< P2P Interface            */
    WICED_ETHERNET_INTERFACE = WWD_ETHERNET_INTERFACE,      /**< Ethernet Interface       */


    WICED_INTERFACE_MAX,       /** DO NOT USE - MUST BE AFTER ALL NORMAL INTERFACES - used for counting interfaces */
    WICED_CONFIG_INTERFACE   = WICED_AP_INTERFACE | (1 << 7), /**< config softAP Interface  */
} wiced_interface_t;
```

Наступний параметр виклику wiced_network_up() – це налаштування мережі, тобто спосіб визначення IP-адреси, мережевої маски, маршрутизатора тощо. Ви можете встановити її статично або використовувати DHCP. WICED-SDK може включити DHCP-сервер всередині вашого пристрою, щоб обслуговувати DHCP-запити з усієї мережі. Це було б корисно, якщо ви виступали в якості точки доступу. Однак для цілей класу ми будемо використовувати "WICED_USE_EXTERNAL_DHCP_SERVER", щоб ви отримали свою IP-інформацію від DHCP, що працює в маршрутизаторі класу. Ось знімок екрана з варіантів:

```
typedef enum
{
    WICED_USE_EXTERNAL_DHCP_SERVER, /**< Client interface: use an external DHCP server
    WICED_USE_STATIC_IP,            /**< Client interface: use a fixed IP address
    WICED_USE_INTERNAL_DHCP_SERVER  /**< softAP interface: use the internal DHCP server
} wiced_network_config_t;
```

Якщо ви використовуєте зовнішній сервер DHCP, вам не потрібно вказувати ip_settings. У цьому випадку просто використовуйте NULL для третього параметра.

Якщо ви не використовуєте зовнішній сервер DHCP, вам потрібно статично вказати параметри IP-мережі, передаючи структуру під назвою wiced_ip_setting_t. Ця структура має три елементи, як це видно тут:

```
/** IP address settings */
typedef struct
{
    wiced_ip_address_t ip_address;  /**< IP address      */
    wiced_ip_address_t gateway;     /**< Gateway address */
    wiced_ip_address_t netmask;     /**< Netmask         */
} wiced_ip_setting_t;
```

7. WICED_RESULT_T

Протягом WICED-SDK повертається значення багатьох функцій, що повідомляє вам про те, що сталося. Повернене значення має тип "wiced_result_t", який є гігантським перерахуванням. Деякі значення, які повертаються, включають WICED_SUCCESS, WICED_PENDING і WICED_ERROR. Якщо ви подивитеся на wiced_result_t, ви не побачите цих значень, тому що перерахунок складається ієрархічно, щоб полегшити його підтримку. Ось це найвищий рівень ієрархії:

```
typedef enum
{
    WICED_RESULT_LIST      ( WICED_             )  /*     0 -   999 */
    WWD_RESULT_LIST        ( WICED_WWD_         )  /*  1000 -  1999 */
    WLAN_RESULT_LIST       ( WICED_WLAN_        )  /*  2000 -  2999 */
    WPS_BESL_RESULT_LIST   ( WICED_BESL_        )  /*  3000 -  3999 */
    RESOURCE_RESULT_LIST   ( WICED_RESOURCE_    )  /*  4000 -  4999 */
    TLS_RESULT_LIST        ( WICED_TLS_         )  /*  5000 -  5999 */
    PLATFORM_RESULT_LIST   ( WICED_PLATFORM_    )  /*  6000 -  6999 */
    TCPIP_RESULT_LIST      ( WICED_TCPIP_       )  /*  7000 -  7999 */
    BT_RESULT_LIST         ( WICED_BT_          )  /*  8000 -  8999 */
    P2P_RESULT_LIST        ( WICED_P2P_         )  /*  9000 -  9999 */
    FILESYSTEM_RESULT_LIST( WICED_FILESYSTEM_ )  /* 10000 - 10999 */
} wiced_result_t;
```

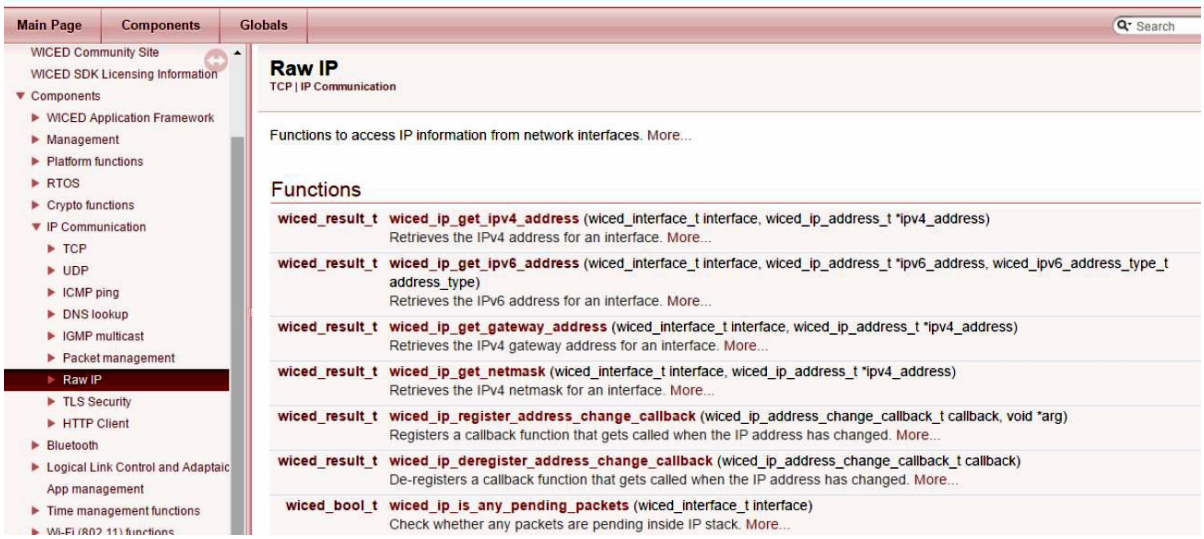
Ви можете переглянути підзапис, клацнувши його правою кнопкою миші. Тобто, якщо натиснути на WICED_RESULT_LIST, ви побачите усі перелічення форми "WICED_". Отже, для успішної команди ви побачите "WICED_SUCCESS", який має значення 0.

```
#define WICED_RESULT_LIST( prefix ) \
    RESULT_ENUM( prefix, SUCCESS,                      0 ),   /**< Success */                      \
    RESULT_ENUM( prefix, PENDING,                      1 ),   /**< Pending */                      \
    RESULT_ENUM( prefix, TIMEOUT,                      2 ),   /**< Timeout */                      \
    RESULT_ENUM( prefix, PARTIAL_RESULTS,              3 ),   /**< Partial results */              \
    RESULT_ENUM( prefix, ERROR,                        4 ),   /**< Error */                        \
    RESULT_ENUM( prefix, BADARG,                       5 ),   /**< Bad Arguments */                \
    RESULT_ENUM( prefix, BADOPTION,                    6 ),   /**< Mode not supported */           \
    RESULT_ENUM( prefix, UNSUPPORTED,                  7 ),   /**< Unsupported function */         \
    RESULT_ENUM( prefix, OUT_OF_HEAP_SPACE,            8 ),   /**< Dynamic memory space exhausted */ \
    RESULT_ENUM( prefix, NOTUP,                        9 ),   /**< Interface is not currently Up */ \
    RESULT_ENUM( prefix, UNFINISHED,                  10 ),   /**< Operation not finished yet */   \
    RESULT_ENUM( prefix, CONNECTION_LOST,             11 ),   /**< Connection to server lost */    \
    RESULT_ENUM( prefix, NOT_FOUND,                   12 ),   /**< Item not found */               \
    RESULT_ENUM( prefix, PACKET_BUFFER_CORRUPT,       13 ),   /**< Packet buffer corrupted */      \
    RESULT_ENUM( prefix, ROUTING_ERROR,               14 ),   /**< Routing error */                \
    RESULT_ENUM( prefix, BADVALUE,                    15 ),   /**< Bad value */                    \
    RESULT_ENUM( prefix, WOULD_BLOCK,                 16 ),   /**< Function would block */         \
    RESULT_ENUM( prefix, ABORTED,                     17 ),   /**< Operation aborted */            \
    RESULT_ENUM( prefix, CONNECTION_RESET,            18 ),   /**< Connection has been reset */    \
    RESULT_ENUM( prefix, CONNECTION_CLOSED,           19 ),   /**< Connection is closed */         \
    RESULT_ENUM( prefix, NOT_CONNECTED,               20 ),   /**< Connection is not connected */  \
    RESULT_ENUM( prefix, ADDRESS_IN_USE,              21 ),   /**< Address is in use */            \
    RESULT_ENUM( prefix, NETWORK_INTERFACE_ERROR,     22 ),   /**< Network interface error */      \
    RESULT_ENUM( prefix, ALREADY_CONNECTED,           23 ),   /**< Socket is already connected */  \
    RESULT_ENUM( prefix, INVALID_INTERFACE,           24 ),   /**< Interface specified in invalid */ \
    RESULT_ENUM( prefix, SOCKET_CREATE_FAIL,          25 ),   /**< Socket creation failed */       \
    RESULT_ENUM( prefix, INVALID_SOCKET,              26 ),   /**< Socket is invalid */            \
    RESULT_ENUM( prefix, CORRUPT_PACKET_BUFFER,       27 ),   /**< Packet buffer is corrupted */   \
    RESULT_ENUM( prefix, UNKNOWN_NETWORK_STACK_ERROR, 28 ),   /**< Unknown network stack error */  \
    RESULT_ENUM( prefix, NO_STORED_AP_IN_DCT,         29 ),   /**< DCT contains no AP credentials */ \
    RESULT_ENUM( prefix, STA_JOIN_FAILED,             30 ),   /**< Join failed */                  \
    RESULT_ENUM( prefix, PACKET_BUFFER_OVERFLOW,      31 ),   /**< Packet buffer overflow */       \
    RESULT_ENUM( prefix, ALREADY_INITIALIZED,         32 ),   /**< Module has already been inited */
```

8. Документація

Відповідна документація щодо функцій управління мережами знаходиться у документації WICED-SDK у розділі Components→Management→Network Management.

Функції, які дозволяють взаємодіяти з мережею IP-файлів, доступні в документації у розділі Components→IP Communication→Raw IP.



Крім того, у каталозі doc є документ під назвою WICED-DCT.pdf, який включає обговорення DCT.

9. Підключення до мережі

Підключення – це метод, який використовується для підключення пристроїв IoT до мережі. Тобто вони повинні знати Wi-Fi SSID, до якого потрібно підключитися, пароль для використання, ключі шифрування для використання тощо. Існує кілька можливих стратегій вирішення цієї проблеми, зокрема наведена нижче.

- Включить у свій пристрій агент Cirrent ZipKey.
  - Агент використовує точку доступу ZipKey для підключення до хмари Cirrent, а потім автоматично налаштовує ваш пристрій IoT для використання вашої мережі Wi-Fi.
  - Хмара Cirrent також надає мережевий інтелект IoT, який дозволяє контролювати, діагностувати та покращувати продуктивність ваших рішень у цій галузі.

- Запустити точку доступу Wi-Fi з веб-сервером на пристрої IoT, а потім підключитися до пристрою IoT з комп'ютера або мобільного телефону. Для цього використовується розділ конфігурації пристрою DCT.
- Підключитися до пристрою IoT за допомогою Bluetooth, а потім за допомогою програми на основі телефону, щоб налаштувати настройки Wi-Fi пристрою.
- Підключити пристрій IoT до комп'ютера за допомогою USB або послідовного з'єднання, а потім налаштувати параметри Wi-Fi пристрою за допомогою комп'ютерного додатку.
- Попередньо запрограмувати пристрій на необхідну інформацію.

10. Порядок роботи

1.Ознайомитись з способом побудови та організацією безпровідних мереж.

2.Переглянути документацію таблиці конфігурації пристрою DCT.

3.Запустити проєкт підключення до наперед визначеної мережі "ww101/05/01_ attach_wpa2". Хід підключення спостерігати в терміналі послідовного порту. Ознайомитись з використаними функціями.

4.Модифікувати попередній проєкт таким чином, щоб під'єднання відбувалось до відкритої мережі WW101OPEN. (WW101OPEN роздайте з телефону наприклад).

5.Запустити проєкт виведення інформації про мережу "ww101/05/03_wpa2_info". Результат роботи спостерігати в терміналі послідовного порту. Пояснити призначення кожного з параметрів. Ознайомитись з використаними функціями.

6.Запустити проєкт переключення між мережами "ww101/05/04_network_switch". Результат роботи спостерігати в терміналі послідовного порту. Пояснити призначення кожного з параметрів. Ознайомитись з використаними функціями.

11. Зміст звіту

1.Написати назву та мету виконання лабораторної роботи.

2.Описати хід виконання лабораторної роботи.

3.Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.

4.Написати висновок після виконання цієї лабораторної роботи. Вказати основні моменти.

12. Контрольні питання

1. Які дані потрібні для організації безпровідної мережі WI-FI?
2. Для яких цілей призначена таблиця конфігурації пристрою (DCT)?
3. Яку роль при організації безпровідного зв'язку WI-FI виконує WICED Wi-Fi SDK?
4. Які варіанти підключення пристрою до мережі WI-FI?

13. Використані джерела

1. Середовище – https://www.cypress.com/products/wiced-software.
2. Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.
3. Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.
4. Відлагоджувальна плата – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

Лабораторна робота № 6

ВСТАНОВЛЕННЯ КОМУНІКАЦІЇ ЗА ДОПОМОГОЮ ПОРТІВ TCP/IP

Мета роботи: зрозуміти як використовувати WICED-SDK для надсилання та отримання даних за допомогою сокетів TCP/IP.

1. Порти – основи зв'язку TCP

Для додатків, наприклад веб-браузера, для зв'язку через транспортний рівень TCP їм потрібно відкрити Socket. Сокет або, точніше, TCP Socket – це просто надійний, упорядкований канал між двома пристроями в Інтернеті. Щоб відкрити сокет, потрібно вказати IP-адресу та номер порту (лише 16-розрядне ціле без підпису) на сервері, з яким ви намагаєтесь поговорити. На сервері працює програма, яка слухає на цьому порту проходження байтів. Сокети однозначно ідентифікуються двома сутностями (IP відправника : вихідний порт) та (IP отримувача : порт призначення), наприклад 192.168.15.8:40287 + 184.27.235.114:80. Це одна з причин, чому може бути кілька відкритих підключень до вебсервера, що працює на порту 80. Локальний (або ефемерний порт) виділяється стеком TCP, а нові порти виділяються на ініціатора (клієнта) для кожного підключення до приймача (сервер).

Існує маса стандартних портів (які ви можете розпізнати) для додатків, зокрема:

- HTTP 80;
- HTTPS 443;
- SMTP 25;
- DNS 53;
- POP 110;
- MQTT 1883.

Зазвичай вони називаються "добре відомими портами" та керуються Інтернетом, що присвоює номер IETF (IANA); IANA гарантує, що жодне з двох програм, розроблених для Інтернету, не використовує один і той же порт (незалежно від UDP або TCP).

WICED легко підтримує TCP-сокети (wiced_tcp_create_socket()), і ви можете створити власний протокол для розмови між вашим пристроєм IoT і сервером, або ви можете реалізувати протокол, визначений кимось іншим.

Наприклад, щоб створити спеціальний протокол, ми можемо визначити WICED-протокол Wi-Fi-протоколу (WWEP) як текстовий протокол ASCII. І клієнт, і сервер надсилають рядок символів, що мають форму:

- Команда: 1 символ, що представляє команду (R = Прочитати, W = Записати, A = Прийнято, X = Не вдалося).
- Ідентифікатор пристрою: 4 символи, що представляють шістнадцяткове значення пристрою, наприклад 1FAE або 002F. Кожен пристрій матиме свій власний унікальний набір реєстру на сервері, тому вам слід використовувати унікальний ідентифікатор (якщо ви не хочете читати/записувати той самий набір реєстру, що й інший пристрій).
- Регістр: 2 символи, що представляють регістр (кожен пристрій має 256 регістрів), наприклад 0F або 1B.

- Значення: 4 символи, що представляють 16-бітний unsigned integer. Значення віддаються за командою "R".

Клієнт може надсилати команди "R" і "W". Сервер відповідає "A" (а також дані, котрі отримав) або "X" (нічого іншого). Сервер містить базу даних, яка буде зберігати записані в нього значення (коли клієнт використовує команду "W") і відправлятиме запитувані значення (коли клієнт використовує команду "R"). Сервер відстежує окремий набір 256 регістрів для кожного ідентифікатора пристрою. Наприклад, зареєструвати з адресою 0×0F для пристрою з ідентифікатором 0×1234 – це не те саме, що зареєструвати з адресою 0×0F для пристрою з ідентифікатором 0×ABCD.

Відкрита версія протоколу працює на порту 27708, а захищена версія TLS працює на порту 40508. У цій роботі ми будемо використовувати в основному відкриту версію протоколу.

Деякі приклади:

- "W0FAC0B1234" запише значення 0×1234 для реєстрації 0×0B для пристрою з ідентифікатором 0×0FAC. Потім сервер відповість "A0FAC0B1234".
- "W01234" є неправильним пакетом, і сервер відповість "X".
- "R0FAC0B" – зчитування регістра 0×0B для ідентифікатора пристрою з ідентифікатором 0×0FAC ". У цьому випадку сервер відповів би "A0FAC0B1234" (значення 1234 було записано в першому випадку).
- "R0BAC0B" – це правильне зчитування, але на цьому пристрої не було записано даних, тож сервер відповість "X".

Зауважте, що "raw" сокети по суті не мають безпеки. TCP-сокет просто надсилає будь-які дані, які були надані через посилання. Це відповідальність за рівень над TCP, такий як SSL або TLS, за шифрування/розшифрування даних, якщо використовується захист.

Сокети доступні у WICED-SDK і дозволяють створити власний протокол. Однак, як правило, розробники в основному використовують один із стандартних протоколів додатків (HTTP, MQTT тощо).

2. WICED-SDK TCP-сервер і клієнт за допомогою Sockets

У наведених нижче прикладах використаємо протокол WWEP, визначений раніше, щоб продемонструвати кроки для створення з'єднання між клієнтом WWEP (198.51.100.14) та сервером WWEP (198.51.100.3) за допомогою сокетів.

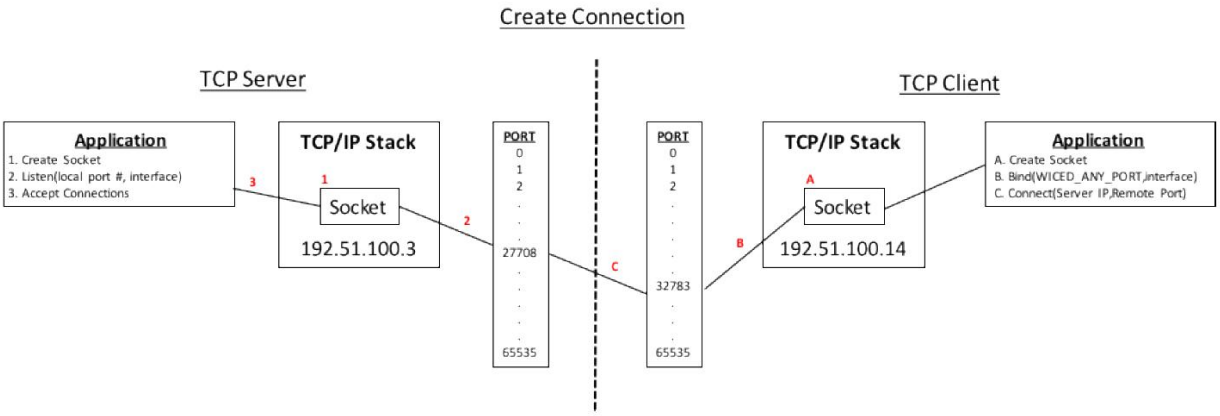


Рис. 1. Демонстрація процесу утворення зв'язку

На рис. 1 описані кроки, необхідні для встановлення TCP-з'єднання між двома пристроями, сервером TCP (зліва від пунктирної лінії) та клієнтом TCP (праворуч). Ці два пристрої вже підключені до мережі IP та їм призначено IP-адреси (192.51.100.3 та 14). У кожній системі є 4 частини.

- Вбудоване програмне забезпечення. Це прошивка, яку пишемо за допомогою WICED-SDK. Існує прошивка як для сервера, так і для клієнта.
- Стек TCP/IP, який обробляє всю комунікацію з мережею.
- Порт, який представляє 65536 TCP-портів (нумеровано 0-65535).
- Буфер пакетів, який представляє оперативну пам'ять 4×~1500 байтів, де зберігаються пакети "T" та "R", що приймають.

Щоб налаштувати з'єднання з сервером TCP, прошивка сервера має бути такою:

1. Створити сокет TCP, викликавши (сокет – це структура типу wiced_tcp_socket_t):
wiced_tcp_create_socket(&socket, WICED_STA_INTERFACE).
2. Послухати сокет на сервері WWEP TCP-порт 27708, викликавши:
wiced_tcp_listen(&socket, 27708); // 27708 is the port number WWEP.
3. Очікувати в потоці чекаючи з'єднання, викликавши:
wiced_tcp_accept(&socket).

Щоб налаштувати з'єднання клієнта TCP, прошивка клієнта:

1. Створіть сокет TCP, викликавши:
wiced_tcp_create_socket(&socket, WICED_STA_INTERFACE).
2. "Прив'язати" до якогось порту TCP (не важливо, до якого, тому ми визначаємо WICED_ANY_PORT, який дозволяє стеку TCP/IP вибирати будь-який доступний порт), викликавши:
wiced_tcp_bind(&socket, WICED_ANY_PORT).

Щоб створити фактичне з'єднання з сервером, вам потрібно зробити дві речі:

1. Знайдіть адресу сервера. Це передається у вигляді структури даних WICED типу wiced_ip_address_t. Припустимо, ви визначили структуру цього типу під назвою serverAddress.

Можна ініціалізувати структуру одним із двох способів – статично або за допомогою DNS:

– Щоб ініціалізувати його статично, ви можете використовувати макроси, надані WICED-SDK так:
SET_IPV4_ADDRESS(serverAddress, MAKE_IPV4_ADDRESS(198, 51, 100, 3)).

– Щоб ініціалізувати його, виконуючи пошук DNS, виконайте таке:
wiced_hostname_lookup("wwep.ww101.cypress.com", &serverAddress, 10000, WICED_STA_INTERFACE);

b) Тепер, коли є адреса сервера, здійснюється підключення до порту 27708 через мережу, викликавши wiced_tcp_connect() і чекаючи TIMEOUT кількості мілісекунд для з'єднання. У нашій локальній мережі час очікування може бути малим <1, але в ситуації з WAN час очікування може бути подовжений на кілька секунд:
wiced_tcp_connect(&socket, &serverAddress, 27708, TIMEOUT).

3. Передача та отримання даних за допомогою потоків

Після створення з'єднання ваша програма захоче передати дані між клієнтом і сервером. Найпростіший спосіб передачі даних через TCP – це використання функцій потоку з WICED-SDK. Функції потоку дозволяють надсилати та отримувати довільну кількість даних, не турбуючись про деталі упаковки даних в рівномірні пакети.

Щоб використовувати потік, потрібно спочатку оголосити структуру потоку, а потім ініціалізувати його з сокетом для вашого мережевого з'єднання:

```
wiced_tcp_stream_t stream;
wiced_tcp_stream_init(&stream, &socket).
```

Коли це зроблено, можна просто записати дані за допомогою функції wiced_tcp_stream_write(). Ця функція приймає потік і повідомлення як параметри. Повідомлення – це лише масив символів, який потрібно надіслати. Коли ви закінчите запис у потік, вам потрібно викликати функцію wiced_tcp_stream_flush (), інакше дані не будуть надсилатися, поки не буде готовий повний пакет. Наступний код демонструє написання одного повідомлення:

```
char sendMessage [] = "TEST_MESSAGE";
wiced_tcp_stream_write (& stream, sendMessage, strlen (sendMessage));
wiced_tcp_stream_flush (& stream).
```

Для зчитування даних із потоку використовується функція wiced_tcp_stream_read(). Ця функція приймає параметри потоку та буфера повідомлень. Функція також вимагає вказати максимальну кількість байтів для читання в буфері та тайм-аут. Функція повертає значення wiced_result_t, яке може бути використане для забезпечення успішного читання потоку.

```
result = wiced_tcp_stream_read (& stream, rbuffer, 11, 500).
```

За лаштунками для читання та запису за допомогою потоків використовуються пакети однакового розміру. Функції потоку в WICED-SDK приховують від вас управління всіма цими пакетами, щоб ви могли зосередитись на більш високих рівнях вашої програми. Однак, якщо ви хочете більше контролювати спілкування, ви можете використовувати API WICED-SDK для надсилання та отримання пакетів безпосередньо.

Після того, як ви закінчите потік, вам потрібно викликати функцію deinit, перш ніж її ініціалізувати. Так само сокет потрібно видалити, коли закінчите з ним. Це досить типово для серверних/клієнтських додатків – тобто ви відкриваєте сокет, ініціалізуєте потік, читаєте/записуєте деякі дані, а потім позбавляєтесь від потоку та сокета. Для наступного набору даних створюються новий сокет і потік.

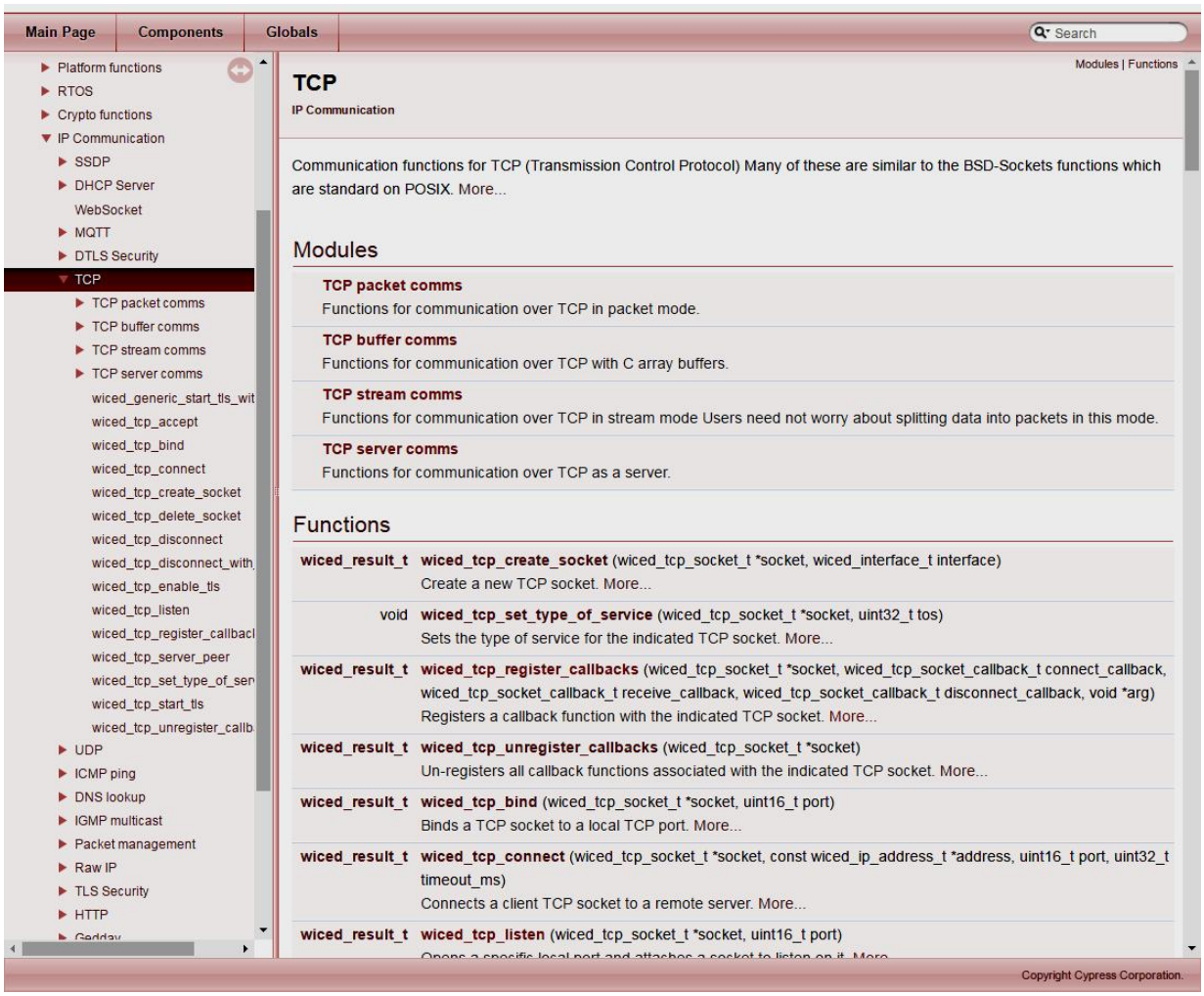
З огляду на сказане вище, прошивка для передачі даних за допомогою потоків може виглядати приблизно так:

```
#define SERVER_PORT (27708)
#define TIMEOUT (2000)
.
wiced_ip_address_t serverAddress;
wiced_tcp_socket_t socket;
wiced_tcp_stream_t stream;
char sendMessage[]="WABCD051234";
.
wiced_hostname_lookup( "wwep.ww101.cypress.com", &serverAddress, 10000,
WICED_STA_INTERFACE );
.
// Loop here for each message to be sent
wiced_tcp_create_socket(&socket, WICED_STA_INTERFACE);
wiced_tcp_bind(&socket, WICED_ANY_PORT );
```

```
wiced_tcp_connect(&socket, &serverAddress, SERVER_PORT, TIMEOUT);
wiced_tcp_stream_init(&stream, &socket);
wiced_tcp_stream_write(&stream, sendMessage, strlen(sendMessage));
wiced_tcp_stream_flush(&stream);
wiced_tcp_stream_deinit(&stream);
wiced_tcp_delete_socket (&socket);
// End of loop
```

4. Документація WICED-Socket

WICED-SDK надає бібліотеку функцій для зв'язку на основі Socket. Документація WICED на розетки міститься в Components -> IP Communication -> TCP. Існують підрозділи для API, специфічні для пакетної комунікації, буферної комунікації, потокової та серверної комунікацій. Ми будемо мати справу переважно з потоковими комунікаціями, але розширені вправи також охоплюватимуть API сокетів та серверів.



5. Виконання завдання

Для демонстрації обміну повідомлень ми не зможемо використати сервер за адресою wwep.ww101.cypress.com, бо він не працює. Необхідно запустити власний сервер на іншій платі відлагодження. Програмний код сервера знаходиться в проєкті "06a/03_server". Його

необхідно модифікувати так, щоб сервер також виконував роль точки доступу. Для цього необхідно:

– у файлі 03_server.c замінити “#define NETWORK_TYPE USE_STA” на стрічку “#define NETWORK_TYPE USE_AP”;

– у файлі 03_server.c у функції void pingAP (wiced_thread_arg_t arg) замінити адресу на 198.51.100.3, оскільки саме таку вказали в структурі вище;

– у файлі wi-fi_config_dct.h вказати назву мережі та пароль до неї. За замовчуванням назва “WW101WPA” та пароль “cypresswicedwi-fi101”.

Після цих змін на сервері запуститься мережа.

В Проєкті клієнта у файлі wi-fi_config_dct.h вказуємо назву мережі сервера та пароль до неї.

Після цього в терміналі клієнта можна спостерігати таке:

```
Joining : WW101WPA

Successfully joined : WW101WPA

Obtaining IPv4 address via DHCP

 L1420 : dhcp_client_init() : DHCP CLIENT hostname = [WICED IP]

IPv4 network ready IP: 198.51.100.4

Setting IPv6 link-local address

IPv6 network ready IP: FE80:0000:0000:0000:4691:60FF:FE28:5F22

DNS Lookup wwep.ww101.cypress.com

Error in resolving DNS using hard coded address

Starting Main Loop

Sent Message=W01DE050000

Sent Message=W01DE050001

Sent Message=W01DE050000

Sent Message=W01DE050001
```

А в терміналі сервера

```
Starting WWEP Server

IPv4 network ready IP: 198.51.100.3

Setting IPv6 link-local address

IPv6 network ready IP: FE80:0000:0000:0000:4491:60FF:FE28:5676

#            IP          Port    Message

------------------------------------------------------------------------

1       198.51.100.5    50142   A01DE050000

2       198.51.100.5    63959   A01DE050001

3       198.51.100.5    57421   A01DE050000

4       198.51.100.5    50711   A01DE050001
```

6. Порядок роботи

1. Ознайомитись з способом надсилання та отримання даних за допомогою сокетів TCP/IP.

2. Запустити на одній платі відлагодження проєкт створення сервера "ww101/06a/03_server". Хід підключення спостерігати в терміналі послідовного порту. Ознайомитись з використаними функціями.

3.Запустити на іншій платі відлагодження проєкт клієнта "ww101/06a/01_client". Хід підключення спостерігати в терміналі послідовного порту. Надсилати дані на сервер. Ознайомитись з використаними функціями.

7. Зміст звіту

1. Написати назву та мету виконання лабораторної роботи.
2. Описати хід виконання лабораторної роботи.
3. Вставити скріншоти послідовності створення проєкту з коротким описом кожного з етапів.
4. Написати висновок після виконання даної лабораторної роботи. Вказати основні моменти.

8. Контрольні питання

1. Як відбувається передача даних за допомогою сокетів?
2. Яка послідовність дій для утворення сервера TCP?
3. Яка послідовність дій клієнта для підєднання до сервера TCP?
4. Для чого використовуються потоки при передачі даних через TCP?

9. Використані джерела

1. Середовище – https://www.cypress.com/products/wiced-software.
2. Файли проєктів https://github.com/cypresssemiconductorco/CypressAcademy_WW101_Files.
3. Відеоуроки – https://www.cypress.com/training/wiced-wi-fi-101-video-tutorial-series.
4. Відлагоджувальна плата – https://www.cypress.com/documentation/development-kitsboards/cyw954907aeval1f-evaluation-kit.

ЗМІСТ

ЕЛЕКТРОННЕ НАВЧАЛЬНЕ ВИДАННЯ

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
ПРОГРАМНІ ЗАСОБИ ІНТЕРНЕТУ РЕЧЕЙ

Навчальний посібник

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

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