УДК 621.383

О.В. Осадчук
В.К. Скощук

ПЕРЕДАЧІ НАКОПИЧЕНИХ ДАНИХ РАДІОТЕХНІЧНОЮ СИСТЕМОЮ НА FPGA ДЛЯ ЧАСТОТНИХ ПЕРЕТВОРЮВАЧІВ ФІЗИЧНИХ ВЕЛИЧИН ВИКОРИСТОВУЮЧИ ТЕХНОЛОГІЮ WIFI

Вінницький національний технічний університет

Анотація

У роботі представлено IoT-систему на базі FPGA та модуля Lilygo LORA32 для збору, обробки та передачі даних у віддалених місцях. Система використовує протоколи LoRa, WiFi і Bluetooth, а формат CBOR забезпечує компактність і швидкість передачі даних. Реалізовано LoRa-концентратор для об'єднання вимірювань і сервер із REST API на базі FastAPI для обробки й зберігання даних. Система характеризується точністю, енергоефективністю й універсальністю, придатною для екологічного моніторингу, автоматизації та інших IoT-сценаріїв.

Ключові слова: IoT, FPGA; ESP32; LoRa; ПЛІС; багатоканальна радіовимірювальна система; частотний перетворювач;транзисторна структура з від'ємним опором

Abstract

The work presents an IoT system based on FPGA and the Lilygo LORA32 module for data collection, processing, and transmission in remote locations. The system utilizes LoRa, WiFi, and Bluetooth protocols, with the CBOR format ensuring data compactness and transmission speed. A LoRa hub is implemented for measurement aggregation, along with a server featuring a REST API built on FastAPI for data processing and storage. The system is characterized by its accuracy, energy efficiency, and versatility, making it suitable for environmental monitoring, automation, and other IoT scenarios.

Keywords: IoT, ESP32; LoRa; FPGA; multi-channel radiomeasuring system; frequency transducer; transistor structure with negativeresistance

Вступ

Сучасні технології вимагають створення точних, енергоефективних вимірювальних систем, здатних працювати у важкодоступних місцях, як-от гірські райони чи промислові зони. У галузях IoT, автоматизації та екологічного моніторингу потрібні багатоканальні системи для паралельного збору й передачі даних у реальному часі. Провідні рішення часто обмежують мобільність і масштабованість, а інтеграція платформ FPGA із бездротовими модулями (LoRa, WiFi, Bluetooth) потребує оптимізації алгоритмів обробки й передачі даних.

Ефективні системи повинні відповідати ключовим вимогам: точність вимірювань, енергоефективність для роботи у віддалених місцях, масштабованість для підтримки численних пристроїв і гнучкість у виборі технологій зв'язку. Запропонована IoT-система на основі FPGA та LoRa вирішує ці виклики. Інтеграція формату CBOR забезпечує компактність і швидкість передачі, а LoRa-концентратор спрощує обробку й доставку даних на сервери або мобільні пристрої. Ця розробка відкриває нові можливості для точних, мобільних і економічних IoT-рішень у різних галузях.

Результати розробки та дослідження

Першим кроком реалізації являється забезпечення доступу концентратора до інтернету, для можливості передачі накопичених даних. Для цього було розширено реалізацію концентратора і додано підтримку передачі даних на вказаний сервер через WiFi. Створено простий інтерфейс який дозволяє:

1) Під'єднатись до WiFi точки доступу.
2) Встановити безпечне з'єднання із сервером.
3) Надсилати вимірювання на сервер.

```
namespace ServerConn {
    bool begin();
    bool isConnected();
    void sendData(const std::vector<uint8_t> &data);
```

```
}
```

При виклику `ServerConn::begin()` відбувається ініціалізація WiFi модуля і під'єднання до вказаної точки доступу, також відбувається налаштування автоматичного перепід'єднання до точки доступу після втрати зв'язку.

```
WiFi.begin(WIFI_SSID.c_str(), WIFI_PASSWORD.c_str());
WiFi.setAutoReconnect(true);
```

Створюється черга розміром 50 повідомлень, яка використовується для збереження вимірювань, що будуть надіслані серверу.

```
 _payloadQueue = xQueueCreate(PAYLOAD_QUEUE_SIZE, sizeof(Payload));
```

Останнім етапом у функції `ServerConn::begin()` являється створення окремого потоку на ядрі 0, що дозволяє паралельне отримання вимірювань через LoRa протокол і передачу цих вимірювань на сервер, без взаємного блокування.

```
 xTaskCreatePinnedToCore(serverTask, "server", 10000, NULL, 5, &_serverTaskHandle, 0);
```

Для кращого розуміння, черга `_payloadQueue` створюється для управління передачею даних між різними частинами програми, зокрема між частиною, яка отримує вимірювання, і частиною, яка їх відправляє на сервер. З архітектурної точки зору, використання черги має кілька переваг:

1) Асинхронність: Черга дозволяє відокремити процес отримання даних від процесу їх відправки. Це означає, що програма може продовжувати виконувати інші завдання, поки дані очікують на відправку в черзі.
2) Буферизація: Черга діє як буфер, що дозволяє накопичувати дані, які ще не були відправлені. Це особливо корисно, якщо швидкість генерації даних перевищує швидкість їх відправки.
3) Потокобезпека: Використання черги забезпечує безпечний спосіб обміну даними між різними потоками у багатопотоковому середовищі, такому як FreeRTOS [1].

Виклик функції `ServerConn::isConnected()` повертає стан під'єднання до WiFi точки доступу.

```
WiFi.status() == WL_CONNECTED
```

Виклик функції `ServerConn::sendData()` створює копію даних які необхідно відправити і записує їх у чергу, де вони будуть оброблені потоком "server", при першій можливості.

```
Payload payload;
payload.data = new uint8_t[data.size()];
payload.size = data.size();
memcpy(payload.data, data.data(), data.size());

if (xQueueSend(_payloadQueue, &payload, PAYLOAD_QUEUE_TIMEOUT) != pdPASS) {
    LOGE(TAG, "Failed to send payload to queue");
    delete[] payload.data;
    return;
}
```

На цьому опис інтерфейсу `ServerConn` завершено і можна перейти до опису внутрішньої реалізації, а саме функції `serverTask()`.Функція `serverTask()` є ключовою частиною реалізації IoT-системи, яка відповідає за обробку черги з даними вимірювань `_payloadQueue` і їхню передачу на сервер через протокол HTTPS [2]. Ця функція виконується у виділеному потоці, що дозволяє забезпечити асинхронність та ефективність передачі даних без блокування інших процесів.

На кожній ітерації перевіряється стан підключення до WiFi та наявність даних у черзі. Якщо підключення до WiFi відсутнє або черга порожня, функція робить паузу на 1 мс, дозволяючи іншим потокам продовжити роботу.

```
if (!isWifiConnected() || xQueueReceive(_payloadQueue, &payload, 0) != pdPASS) {
    vTaskDelay(1);
    continue;
}
```

Після отримання даних з черги ініціалізується `WiFiClientSecure`, налаштовуючи сертифікат довіреності `CA_CERT`, необхідний для встановлення захищеного з'єднання.

```
WiFiClientSecure client;
client.setCACert(CA_CERT);
```

Встановлюється з'єднання із сервером через HTTPS. Для цього використовується шлях `"https://server_ip:8080/device/data"`, який включає IP-адресу сервера, порт і маршрут до API. До HTTP-запиту додається заголовок "`Content-Type: application/octet-stream`", що вказує на двійковий формат переданих даних. Дані передаються методом POST [3]. Код відповіді сервера зберігається у змінній `httpResponseCode`, і, залежно від результату, виводиться відповідне повідомлення у термінал.

```
HTTPClient https;
if (https.begin(SERVER_DATA_PATH.c_str())) {
    // Add header
    https.addHeader("Content-Type", "application/octet-stream");
    // Send payload
    int httpResponseCode = https.POST(payload.data, payload.size);
    LOGI(TAG, "HTTP Response: %d", httpResponseCode);
    if (httpResponseCode == HTTP_CODE_OK) LOGI(TAG, "POST Success");
    else LOGE(TAG, "POST Failed, Error: %d (%s)", httpResponseCode,
https.errorToString(httpResponseCode).c_str());
    // Close connection
    https.end();
}
```

Після завершення передачі звільняється пам'ять, виділена для збереження даних у черзі.

```
delete[] payload.data;
```

Функція `serverTask()` відіграє ключову роль у забезпеченні надійної та захищеної передачі даних у IoT-системі, роблячи її адаптованою до вимог сучасних мережевих рішень. Для забезпечення безпеки передачі даних у системі використовується протокол HTTPS (Hypertext Transfer Protocol Secure**)**. Основні аспекти безпеки реалізовані за рахунок:

1. Шифрування даних. Дані, що передаються між пристроєм і сервером, шифруються за допомогою SSL/TLS. Це забезпечує конфіденційність і захищає дані від перехоплення зловмисниками.
2. Перевірка сертифікатів. У системі використовується сертифікат довіреності (CA_CERT), який гарантує автентичність сервера. Це дозволяє переконатися, що дані передаються саме на легітимний сервер, а не на підроблений вузол.
3. Цілісність даних. Протокол HTTPS включає механізми перевірки цілісності, які гарантують, що передані дані не були змінені під час їх транспортування.
4. Двостороння аутентифікація. Сертифікат сервера використовується для підтвердження його автентичності. У разі потреби система може бути доповнена механізмами клієнтської аутентифікації для ще більш високого рівня захисту.
5. Захист від атак MITM [4]. Використання захищеного протоколу з перевіркою сертифікатів значно знижує ризик атак, при яких зловмисник може перехопити або змінити дані.

Ці заходи забезпечують надійну та захищену передачу вимірювань, зберігаючи конфіденційність, цілісність і автентичність інформації в системі.

Наступним етапом являється налаштування сервера і реалізація REST API. Для цього було:

1) Орендовано хмарні обчислення на платформі AWS [5].
2) Створено *Instance* типу *t2.micro* із операційною системою Linux [6].
3) Надано доступ до порту 8080 для вхідних запитів, який буде використовуватись для HTTP сервером.
4) Створено сертифікат який буде використаний для встановлення безпечного каналу зв'язку між сервером і всіма іншими пристроями: *sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout selfsigned.key -out selfsigned.crt*.

Висновки

У роботі представлено IoT-систему для збору, обробки та передачі вимірювань із багатоканальних радіотехнічних систем на основі FPGA. Основна мета — створення універсальної, енергоефективної системи, здатної працювати у віддалених умовах. Завдяки інтеграції модуля Lilygo LORA32 та протоколів LoRa і WiFi реалізовано багаторівневу архітектуру з високою продуктивністю та гнучкістю. LoRa-концентратор забезпечує збір і передачу даних на сервер через WiFi, що гарантує високу швидкість обміну. Сервер із REST API на базі FastAPI приймає пакети у форматі CBOR, зберігає дані у SQLite та надає доступ через API-інтерфейси. Захист даних забезпечено протоколом HTTPS.

Система відзначається точністю, енергоефективністю та універсальністю, підходячи для промислової автоматизації, моніторингу довкілля та IoT-застосувань. Подальший розвиток платформи передбачає вдосконалення обробки даних, розширення пропускної здатності та впровадження нових функцій для реального часу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Barry, R. Mastering the FreeRTOS Real Time Kernel – A Hands-On Tutorial Guide. Real Time Engineers Ltd., 2016. – 144 pages.
2. Rescorla, E. HTTP Over TLS. RFC 2818, May 2000. – 7 pages. [Електронний ресурс]: - Режим доступу: https://www.rfc-editor.org/rfc/rfc2818
3. Fielding, R., & Reschke, J. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, June 2014. – 133 pages.
4. Rescorla, E., & Korver, B. Guidelines for Writing RFC Text on Security Considerations. RFC 3552, July 2003. – 44 pages.
5. Amazon Web Services. Overview of Amazon Web Services. AWS Whitepaper, 2023. – 101 pages. [Електронний ресурс]: - Режим доступу: docs.aws.amazon.com
6. Corbet, J., Rubini, A., & Kroah-Hartman, G. Linux Device Drivers. 3rd ed., O'Reilly Media, 2005. – 636 pages.

Осадчук Олександр Володимирович — докт. техн. наук, проф., зав. кафедри інформаційних радіоелектронних технологій і систем, Вінницькийнаціональний технічний університет, osadchuk.av69@gmail.com

Скощук Валентин Костянтинович — аспірант кафедри інформаційних радіоелектронних технологій і систем, Вінницький національний технічнийуніверситет, skoschuk999@gmail.com

Alexander Osadchuk — Doc. Tech. Sc., prof. Head of Department of Information Radio Engineering Technologies and Systems, Vinnytsia NationalTechnical University, Vinnytsia, Ukraine, osadchuk.av69@gmail.com

Valentyn Skoshchuk – graduate student of the Department of Information Radio Engineering Technologies and Systems, Vinnytsia NationalTechnical University, Vinnytsia, Ukraine, skoschuk999@gmail.com