DOI 10.31891/2307-5732-2025-347-51
УДК 621.382

ОСАДЧУК ОЛЕКСАНДР
Вінницький національний технічний університет
https://orcid.org/0000-0001-6662-9141
e-mail: osadchuk.av69@gmail.com
СКОЩУК ВАЛЕНТИН
Вінницький національний технічний університет
https://orcid.org/0009-0008-9762-2397
e-mail: skoschuk999@gmail.com

БЕЗДРОТОВА СИСТЕМА ЗБОРУ ДАНИХ НА FPGA ДЛЯ ЧАСТОТНИХ ПЕРЕТВОРЮВАЧІВ ФІЗИЧНИХ ВЕЛИЧИН ІЗ ВИКОРИСТАННЯМ LORA

У роботі представлено систему збору та передачі вимірювань із багатоканальної радіотехнічної системи на основі FPGA для частотних перетворювачів фізичних величин із використанням протоколу LoRa. Для реалізації цієї системи було завершено розробку пристрою збору вимірювань та створено концентратор, здатний приймати дані від кількох таких пристроїв. Концентратор побудовано на базі модуля Lilygo LORA32, що дозволяє накопичувати вимірювання в єдиній точці з подальшою передачею через бездротові канали зв'язку, зокрема WiFi та Bluetooth. Запропонований підхід базується на використанні модуля Lilygo LORA32 для створення багатоканальної вимірювальної системи, що поєднує переваги високопродуктивних обчислювальних платформ із можливостями бездротової передачі даних. Основною метою розробки є забезпечення надійного збору, обробки та передачі вимірювань у системах із частотними сенсорами фізичних величин. Цього досягнуто завдяки розробленому протоколу передачі даних у компактному форматі CBOR та інтеграції технології LoRa. Розроблена система стала важливим кроком у створенні сучасних вимірювальних платформ, що вирізняються високою точністю, надійністю та можливістю бездротової передачі даних. Подальший розвиток платформи передбачає впровадження нових методів обробки даних, підвищення продуктивності концентратора та створення інтегрованих рішень для моніторингу в реальному часі. Це відкриває перспективи для широкого використання системи в IoT, автоматизації виробництва, моніторингу довкілля та багатьох інших галузях.

Ключові слова: LILYGO LoRa32, LoRa, WiFi, Bluetooth, NIOS II, FPGA, багатоканальний частотомір, сенсор з частотним виходом, радіовимірювальні перетворювачі фізичних величин, частота.

OSADCHUK OLEKSANDR, SKOSHCHUK VALENTYN
Vinnytsia National Technical University

WIRELESS DATA COLLECTION SYSTEM ON FPGA FOR FREQUENCY CONVERTERS OF PHYSICAL QUANTITIES USING LORA

The study presents a system for collecting and transmitting measurements from a multichannel radio engineering system based on FPGA for frequency converters of physical quantities using the LoRa protocol. To implement this system, the development of a measurement collection device was completed, and a hub capable of receiving data from several such devices was created. The hub is built on the Lilygo LORA32 module, enabling the accumulation of measurements in a single point with subsequent transmission via wireless communication channels, including WiFi and Bluetooth. The proposed approach is based on the use of the Lilygo LORA32 module to create a multichannel measurement system that combines the advantages of high-performance computing platforms with wireless data transmission capabilities. The primary goal of the development is to ensure reliable collection, processing, and transmission of measurements in systems with frequency sensors of physical quantities. This was achieved through the development of a data transmission protocol in a compact CBOR format and the integration of LoRa technology. The developed system represents an important step toward the creation of modern measurement platforms characterized by high accuracy, reliability, and wireless data transmission capabilities. Further development of the platform involves the implementation of new data processing methods, improved hub performance, and the creation of integrated solutions for real-time monitoring. This opens up opportunities for the broad application of the system in IoT, industrial automation, environmental monitoring, and many other fields.

Keywords: LILYGO LoRa32, NIOS II, FPGA, multichannel frequency meter, sensor with frequency output, radio measuring transducers of physical quantities, frequency.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У сучасному світі стрімко зростає потреба у високоточних вимірюваннях, особливо у сферах моніторингу навколишнього середовища, промислової автоматизації та Інтернету речей (IoT). Одним із ключових викликів у таких системах є забезпечення ефективного збору, обробки та передачі великого обсягу вимірювальних даних із мінімальними витратами енергії та ресурсів. Наразі традиційні підходи до реалізації вимірювальних систем переважно базуються на проводових технологіях, що обмежує їх мобільність і здатність до масштабування. Це ускладнює застосування таких систем у розподілених структурах або важкодоступних місцях. З появою технологій FPGA [1-4] відкрилися нові можливості для створення інтегрованих систем завдяки їхній гнучкості та високій продуктивності. Сучасні мікропроцесорні ядра, такі як NIOS II [6-10], дозволяють розробляти універсальні та масштабовані обчислювальні системи без необхідності детального знання апаратних особливостей.

Проте інтеграція FPGA із бездротовими комунікаційними модулями, такими як LoRa, WiFi та Bluetooth, залишається складним завданням, що потребує значних зусиль в оптимізації та адаптації. У цьому контексті запропонована система на базі FPGA та модуля Lilygo LORA32 вирішує важливу практичну задачу: забезпечення паралельного вимірювання та бездротової передачі даних у багатоканальних системах.

Особливістю запропонованого підходу є використання формату CBOR для створення компактних даних, що знижує затримки під час передачі. Такий підхід дозволяє створювати універсальні

рішення для збору та обміну інформацією навіть у складних умовах експлуатації, зокрема для мобільних пристроїв та віддалених мереж. Завдяки технології LoRa система забезпечує високу енергоефективність і значну дальність передачі даних, що є критично важливим для багатьох сфер застосування. Поєднання сучасних підходів до вимірювання, таких як використання FPGA з високим рівнем паралелізму обчислень, із можливостями бездротових модулів відкриває нові перспективи для підвищення продуктивності, адаптивності та масштабності вимірювальних систем.

Аналіз останніх досліджень

У роботах [2–4] наведено приклад розробки багатоканальної системи вимірювання частоти на основі FPGA та поступового розширення її функціоналу. Для реалізації частотоміра використовувалася мікросхема Altera Cyclone IV EP4CE10F17C8. У систему інтегровано мікропроцесорне ядро NIOS II. Ця інтеграція дозволила зробити систему гнучкішою, додати функції попередньої обробки та фільтрації отриманих даних, а також змінювати кількість частотомірів без необхідності модифікації алгоритму обробки даних. Система також оснащена інтерфейсом для взаємодії із цифровими сенсорами через шину I2C. Для цього в ядрі NIOS II була додана підтримка I2C, реалізована у вигляді апаратного блоку, а також розроблено програмне забезпечення для забезпечення роботи цієї шини. На рис. 1 представлено блок-схему кінцевої системи. Основний функціонал системи включає вимірювання частоти з вхідних каналів, отримання даних із цифрових сенсорів, формування пакету з виміряними даними та їх надсилання через UART.

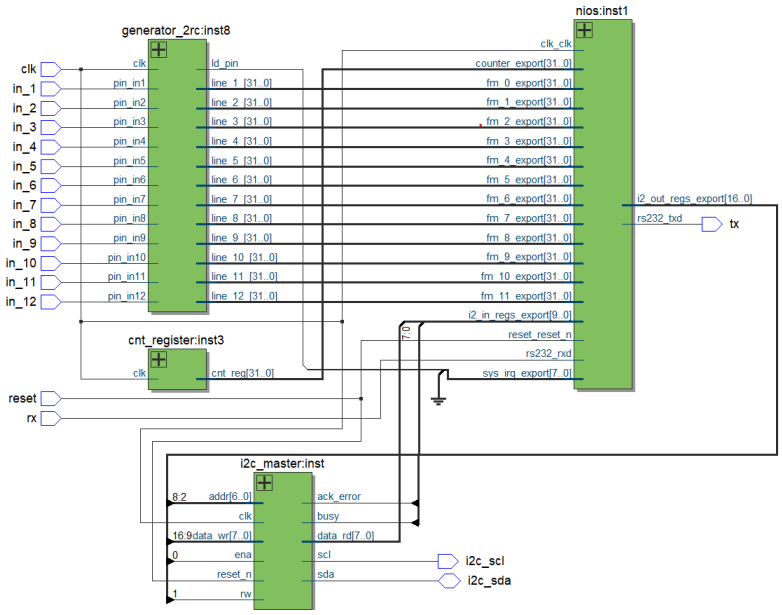

Рис. 1. Схема багатоканального частотоміра з використанням ядра NIOS II і підтримкою I2C протоколу

Результати роботи розробленої системи продемонстровані під час використання генератора частоти на 85 кГц та цифрових сенсорів. Перші 12 значень у результатах відповідають показникам частотомірів, наступне значення — температурі, ще три — показникам акселерометра, а останні три — даним гіроскопа, рис. 2.

```
[1000] 84330 0 0 0 0 0 0 0 0 0 0 0 21800 9790 -15 15 -27 13 0
[1100] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9710 -29 164 -27 13 3
[1200] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -31 165 -27 13 6
[1300] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9740 -31 165 -27 13 6
[1400] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -29 164 -27 13 3
[1500] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -15 15 -27 13 0
[1600] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -31 165 -27 13 6
[1700] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -29 164 -27 13 3
[1800] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -15 15 -27 13 0
[1900] 84340 0 0 0 0 0 0 0 0 0 0 0 21800 9760 -29 164 -27 13 3
```

Рис. 2. Приклад виміряних даних

У роботі [5] представлено прототип пристрою збору вимірювань, розроблений на базі модуля Lilygo LORA32 для багатоканальної радіотехнічної системи на основі FPGA, орієнтованої на частотні перетворювачі фізичних величин. Основною метою була забезпечена можливість попередньої обробки та бездротової передачі даних через канал зв'язку LoRa. Було налаштовано середовище розробки за допомогою VS Code та PlatformIO, створено драйвер UART для зчитування посилок, реалізовано алгоритми парсингу отриманих даних, їх обробки та перевірки на помилки, а також розроблено методи формування компактних посилок для подальшої передачі.

Формулювання цілей статті

Метою роботи є: створення системи збору та передачі вимірювань із багатоканальної радіотехнічної системи на основі FPGA для частотних перетворювачів фізичних величин [4], із

використанням протоколу LoRa. Для досягнення цієї мети буде завершено розробку пристрою збору вимірювань [5], а також створено концентратор для отримання даних із масиву таких пристроїв. Концентратор буде реалізовано на базі модуля Lilygo LORA32 [11], що забезпечить можливість збору та накопичення вимірювань в одній точці. Це дозволить передавати зібрані дані через бездротові канали зв'язку, такі як WiFi і Bluetooth.

Теоретичні та експериментальні дослідження

Для реалізації системи збору та передачі вимірювань (рис. 3) необхідно завершити розробку пристрою збору вимірювань [6]. Це включає зміну алгоритму формування пакета даних та створення бібліотеки для передачі виміряних даних через протокол LoRa. Наступним етапом є розробка прототипу концентратора, здатного одночасно отримувати дані від кількох вимірювальних приладів.

Новий алгоритм формування пакета даних базується на форматі CBOR [12]. CBOR — це формат даних, який відзначається надзвичайно малим розміром коду, компактністю повідомлень та розширюваністю без необхідності узгодження версій. На рис. 4 зображено структуру нового пакета, який формуватиметься на пристрої збору вимірювань і передаватиметься на концентратор у вигляді корисного навантаження. Пакет містить наступні поля:

1) MAC-адресу пристрою, який згенерував цей пакет.
2) Номер пакета.
3) Час генерації пакета.
4) Дані, зібрані з багатоканальної радіотехнічної системи на основі FPGA для частотних перетворювачів фізичних величин [4].
5) Додаткове поле з інформацією про концентратор, яке додається до пакета на стороні концентратора.

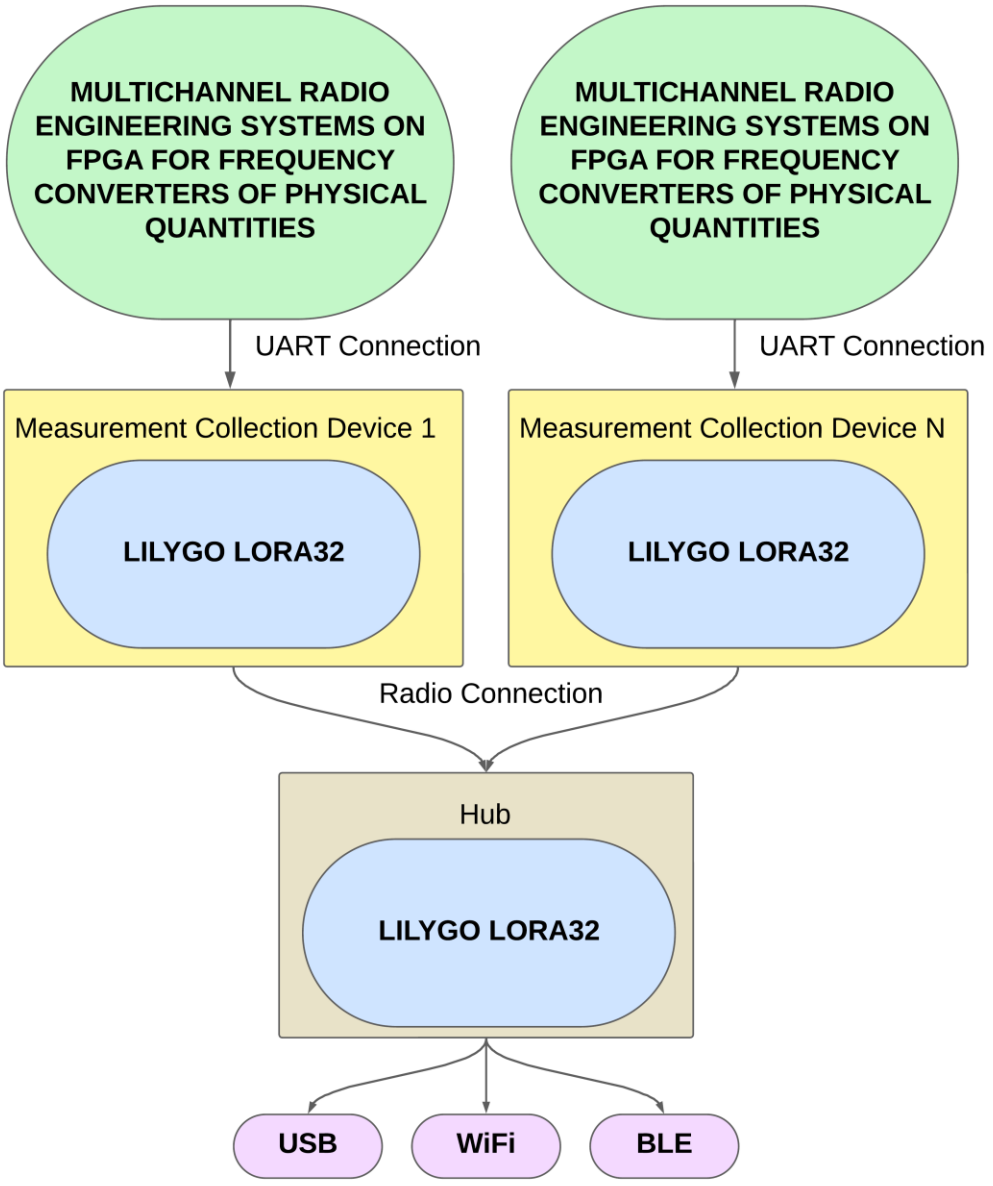

Рис. 3. Блок-схема системи збору та передачі вимірювань за допомогою протоколу LoRa

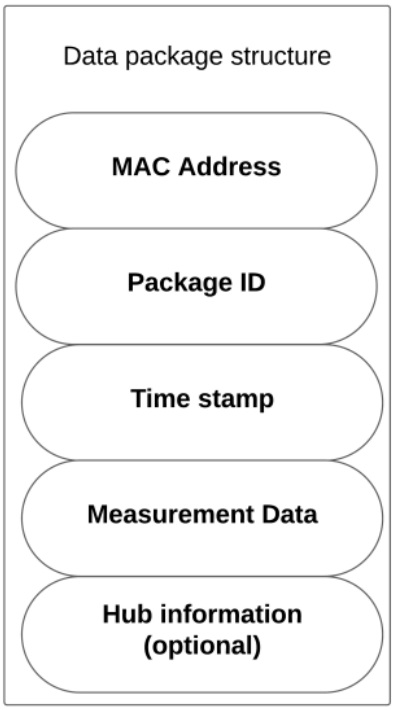

Рис. 4. Структура пакету даних

Для формування описаного пакета створено клас *DataPackage* (рис. 5-6), який реалізує алгоритм конвертування початкового набору даних у пакет і навпаки.

```
enum DataId {
    MAC_ADDR = 0, // 6 bytes array, example: {0x12, 0x34, 0x56, 0x78, 0x90, 0xAB} == 12:34:56:78:90:AB
    PKG_ID = 1, // uint32_t
    TIME_STAMP_MS = 2, // uint64_t
    FM_DATA = 3, // FmData, optional
    HUB_INFO = 4, // HubData, optional
    MAX_DATA_ID
};

class DataPackage {
public:
    DataPackage();
    ~DataPackage();
    bool setRawData(const std::vector<uint8_t> &data);
    std::vector<uint8_t> getRawData();
    bool isMacAddrExist();
    bool setMacAddr(const std::vector<uint8_t> &macAddr);
    std::vector<uint8_t> getMacAddr();
    bool isPkgIdExist();
    bool setPkgId(uint32_t pkgId);
    uint32_t getPkgId();
    bool isTimeStampExist();
    bool setTimeStampMs(uint64_t timeStamp);
    uint64_t getTimeStampMs();
    bool isFmDataExist();
    bool setFmData(const FmData &data);
    FmData getFmData();
private:
    template<typename T>
    bool decodeValue(…);
    bool decodePkgId(…);
    bool decodeMacAddr(…);
    bool decodeTimeStamp(…);
    bool decodeFmData(…);
    void encodeMacAddr(…);
    void encodePkgId(…);
    void encodeTimeStamp(…);
    void encodeFmData(…);
private:
    std::vector<uint8_t> _macAddr;
    uint32_t *_pkgId;
    uint64_t *_timeStamp;
    FmData *_fmData;
};
```

Рис. 5. Інтерфейс для кодування і декодування пакету даних

```
bool DataPackage::decodeFmData(const std::vector<uint8_t> &data, FmData &fmData) {
    CborParser parser;
    CborValue it, map, array;
    bool result = false;
    CborError err = cbor_parser_init(data.data(), data.size(), 0, &parser, &it);
    if (err != CborNoError || !cbor_value_is_map(&it)) return result;
    err = cbor_value_enter_container(&it, &map);
    if (err != CborNoError) return result;
    while (!cbor_value_at_end(&map)) {
        int dataId;
        if (cbor_value_get_int(&map, &dataId) != CborNoError ||
            cbor_value_advance(&map) != CborNoError) break;
        if (dataId == static_cast<int>(DataId::FM_DATA)) {
            if (cbor_value_enter_container(&map, &array) != CborNoError) break;
            for (size_t i = 0; i < FM_FREQS_CNT && !cbor_value_at_end(&array); i++) {
             if (cbor_value_get_int(&array, reinterpret_cast<int32_t*>(&fmData.freqs[i])) !=
CborNoError ||
                    cbor_value_advance(&array) != CborNoError) break;
            }
                   int16_t* fields[] = {&fmData.temp, &fmData.accel_x, &fmData.accel_y,
&fmData.accel_z,
                                        &fmData.gyro_x, &fmData.gyro_y, &fmData.gyro_z};
            for (auto field : fields) {
                if (cbor_value_at_end(&array)) break;
                int tmp;
                if (cbor_value_get_int(&array, &tmp) != CborNoError ||
                    cbor_value_advance(&array) != CborNoError) break;
                *field = static_cast<int16_t>(tmp);
            }
            cbor_value_leave_container(&map, &array);
            result = true;
            break;
        }
        if (cbor_value_advance(&map) != CborNoError) break;
    }
    return result;
}

void DataPackage::encodeFmData(CborEncoder* mapEncoder, const FmData& fmData) {
    cbor_encode_int(mapEncoder, static_cast<int>(DataId::FM_DATA));
    CborEncoder arrayEncoder;
    cbor_encoder_create_array(mapEncoder, &arrayEncoder, FM_FREQS_CNT + 7);
    for (int i = 0; i < FM_FREQS_CNT; i++) {
        cbor_encode_int(&arrayEncoder, fmData.freqs[i]);
    }
    cbor_encode_int(&arrayEncoder, fmData.temp);
    cbor_encode_int(&arrayEncoder, fmData.accel_x);
    cbor_encode_int(&arrayEncoder, fmData.accel_y);
    cbor_encode_int(&arrayEncoder, fmData.accel_z);
    cbor_encode_int(&arrayEncoder, fmData.gyro_x);
    cbor_encode_int(&arrayEncoder, fmData.gyro_y);
    cbor_encode_int(&arrayEncoder, fmData.gyro_z);
    cbor_encoder_close_container(mapEncoder, &arrayEncoder);
}
```

Рис. 6. Приклад кодування і декодування виміряних даних

Під час створення *DataPackage* ключовою вимогою була можливість повторного використання на стороні концентратора. Також цей підхід застосовано до класу *LoraTransport*, який реалізує алгоритм передачі та отримання даних через протокол LoRa. Клас *LoraTransport* побудований на основі бібліотеки *RadioLib* [13]. Першим кроком реалізації *LoraTransport* є визначення структури пакета для бездротової передачі (рис. 7). Пакет містить такі поля:

1) Загальний ідентифікатор пакета для фільтрування пакетів, згенерованих іншими системами.
2) MAC-адресу пристрою, який згенерував цей пакет.
3) Ідентифікатор пакета даних.
4) Кількість частин з яких складається пакет даних.
5) Номер поточної частини.
6) Кількість байт у корисному навантажені.
7) Корисне навантаження.

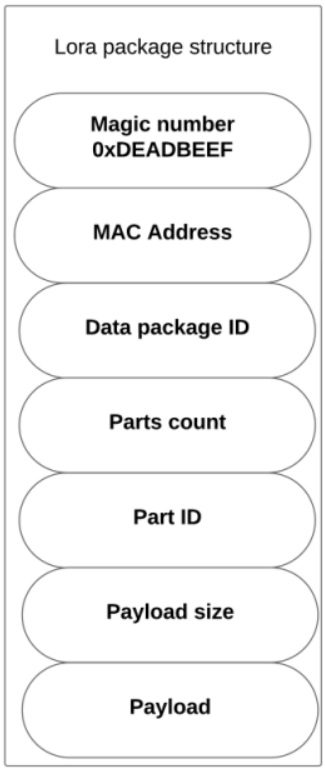

Рис. 7. Структура LoRa пакету

Розмір LoRa-пакета фіксований (рис. 8) і становить 119 байт, із яких 100 байт відводиться під корисне навантаження. Якщо розмір корисного навантаження перевищує 100 байт, дані передаються у кількох пакетах.

```
#pragma pack(push, 1)
struct LoraPkg {
    uint32_t magic; // Magic number to identify the package
    uint8_t srcMacAddr[LORA_PKG_SRC_MAC_ADDR_SIZE]; // Source MAC address
    uint32_t dataPkgId; // Payload package ID
    uint16_t partsCnt; // Total parts count, package can be splitted into multiple parts if
payload is more than LORA_PKG_PAYLOAD_SIZE_BYTES bytes
    uint16_t partId; // Current part ID
    uint8_t payloadSize; // Count of bytes in the payload
    uint8_t payload[LORA_PKG_PAYLOAD_SIZE_BYTES]; // Payload data, always fixed size, even if
payload is less than LORA_PKG_PAYLOAD_SIZE_BYTES, unused bytes are filled with 0
};
#pragma pack(pop)
```

Рис. 8. Опис LoRa-пакету у вигляді структури LoraPkg

Наступним етапом реалізації *LoraTransport* є конфігурація каналу зв'язку в протоколі LoRa. Першою важливою конфігурацією є відсутність додаткового заголовка (Implicit Header) у LoRa-пакеті. Цей заголовок додається модулем SX1276 і обробляється приймальною стороною для визначення параметрів корисного навантаження. Переваги та недоліки режиму Implicit Header:

1) Швидша передача: скорочений час пакета, менше енергоспоживання.
2) Більше корисних даних: вища пропускна здатність.
3) Менша зайнятість каналу: зниження ймовірності колізій.
4) Фіксовані параметри: необхідність попереднього узгодження налаштувань.
5) Більший ризик помилок: менший захист даних без заголовка.
6) Обмеження для SF6: обов'язковий Implicit Header на високій швидкості.

Після налаштування режиму Implicit Header можна перейти до додаткових конфігурацій. Ці налаштування є другорядними за впливом на реалізацію і можуть змінюватися у майбутньому (рис. 9):

1) Центральна частота передачі – 915МГц.
2) Ширина смуги передачі – 500 кГц.
3) Фактор розширення – 6.
4) Рівень корекції помилки – 7.
5) Синхронізаційне слово – 0x99.
6) Довжина преамбули – 20 бітів.
7) Потужність передавача – 10дБм.

```
#define LORA_FREQ_MHZ      915.0
#define LORA_BW_KHZ        500.0
#define LORA_SF            6
#define LORA_CR            7
#define LORA_SYNC_WORD     0x99
#define LORA_POWER_DBM     10
#define LORA_PREAMBLE_LEN  20
#define LORA_GAIN          0

LoraTransport::LoraTransport(const      std::vector<uint8_t>      &macAddr,      ErrorCallback
errorCallback) : _macAddr(macAddr),
    _module(LORA_CS_PIN,      LORA_IO0_PIN,      LORA_RST_PIN,      LORA_IO1_PIN,      _spi,
RADIOLIB_DEFAULT_SPI_SETTINGS),             _radio(&_module),             _mode(Mode::IDLE),
_errorCallback(errorCallback) {
    if (macAddr.size() != LORA_PKG_SRC_MAC_ADDR_SIZE) return;
    _spi.begin(LORA_SCK_PIN, LORA_MISO_PIN, LORA_MOSI_PIN, LORA_CS_PIN);
    pinMode(LORA_IO1_PIN, INPUT);
    pinMode(LORA_IO2_PIN, INPUT);
    uint8_t   status   =   _radio.begin(LORA_FREQ_MHZ,   LORA_BW_KHZ,   LORA_SF,   LORA_CR,
LORA_SYNC_WORD, LORA_POWER_DBM, LORA_PREAMBLE_LEN, LORA_GAIN);
    _radio.implicitHeader(0);
}
```

Рис. 9. Налаштування другорядних параметрів

Після визначення структури пакета і конфігурацій LoRa-каналу необхідно створити інтерфейс користувача для отримання і передачі даних довільного розміру. Інтерфейс (рис. 10) включає асинхронні методи для надсилання й отримання даних, механізм сповіщення про події та можливість визначення поточного стану *LoraTransport*.

```
class LoraTransport {
    struct ReceivedPkg {
        std::vector<uint8_t> srcMacAddr;
        uint32_t dataPkgId;
        uint8_t partId;
        std::vector<uint8_t> data;
    };
public:
    enum Mode {
        IDLE,
        SENDING,
        RECEIVING
    };
    using SendPkgDoneCallback = std::function<void(uint32_t dataPkgId)>;
    using  ReceivedPkgCallback  =  std::function<void(const  std::vector<uint8_t>  &srcMacAddr,
uint32_t dataPkgId, const std::vector<uint8_t> &data)>;
    using ErrorCallback = std::function<void(Mode mode)>;
public:
    LoraTransport(…);
    ~LoraTransport();
    bool sendBlocking(…);
    bool startSending(…);
    bool startReceiving(…);
    void stopReceiving();
    Mode getMode();
    void loop();
private:
    bool send();
    void processReceivedPackage(…);
private:
    SPIClass _spi;
    std::vector<uint8_t> _macAddr;
    Module _module;
    SX1276 _radio;
    Mode _mode;
    SendPkgDoneCallback _sendPkgDoneCallback;
    ReceivedPkgCallback _receivedPkgCallback;
    ErrorCallback _errorCallback;
    uint32_t _sendingPkgId;
    std::vector<uint8_t> _sendingData;
    std::map<uint64_t, ReceivedPkg> _receivedPkgs;
};
```

Рис. 10. Імплементація описаного інтерфейсу

Наведемо приклад коду який ініціює передачу і отримання даних з файлу *LoraTransport.cpp* (рис. 11).

```
bool LoraTransport::startSending(uint32_t dataPkgId, const std::vector<uint8_t> &data,
SendPkgDoneCallback callback) {
    if (_mode != Mode::IDLE) return false;
    if (!data.size()) return false;
    pkgIsSentFlag = false;
    _mode = Mode::SENDING;
    _sendingPkgId = dataPkgId;
    _sendPkgDoneCallback = callback;
    _sendingData = data;
    _radio.setPacketSentAction(pkgIsSent);
    if (!send()) {
        _mode = Mode::IDLE;
        return false;
    }
    return true;
}

bool LoraTransport::startReceiving(ReceivedPkgCallback callback) {
    if (_mode != Mode::IDLE) return false;
    pkgIsReceivedFlag = false;
    _mode = Mode::RECEIVING;
    _receivedPkgCallback = callback;
    _radio.setPacketReceivedAction(pkgIsReceived);
    _radio.setCRC(true);
    uint16_t state = _radio.startReceive(sizeof(LoraPkg), RADIOLIB_SX127X_RXCONTINUOUS);
    if (state != RADIOLIB_ERR_NONE) {
        _mode = Mode::IDLE;
        return false;
    }
    return true;
}
```

Рис. 11. Підготовка до передачі і отримання даних

Код який виконує надсилання даних, рис. 12, файл *LoraTransport.cpp*.

```
bool LoraTransport::send() {
    LoraPkg pkg;
    uint16_t partsCnt = (_sendingData.size() / LORA_PKG_PAYLOAD_SIZE_BYTES) +
(_sendingData.size() % LORA_PKG_PAYLOAD_SIZE_BYTES ? 1 : 0);
    memset(&pkg, 0, sizeof(pkg));
    pkg.magic = LORA_PKG_MAGIC;
    memcpy(pkg.srcMacAddr, _macAddr.data(), LORA_PKG_SRC_MAC_ADDR_SIZE);
    pkg.dataPkgId = _sendingPkgId;
    pkg.partsCnt = partsCnt;
    pkg.partId = 1;
    pkg.payloadSize = partsCnt == 1 ? _sendingData.size() : LORA_PKG_PAYLOAD_SIZE_BYTES;
    memcpy(pkg.payload, _sendingData.data(), pkg.payloadSize);
    uint16_t state = _radio.startTransmit((uint8_t*)&pkg, sizeof(pkg));
    if (state != RADIOLIB_ERR_NONE) return false;
    if (partsCnt != 1) _sendingData.assign(_sendingData.begin() + LORA_PKG_PAYLOAD_SIZE_BYTES,
_sendingData.end());
    else _sendingData.clear();
    return true;
}
```

Рис. 12. Надсилання даних

Код для обробки отриманого LoRa пакету, рис. 13, файл *LoraTransport.cpp.*

```
void LoraTransport::processReceivedPackage(LoraPkg &pkg) {
    if (pkg.magic != LORA_PKG_MAGIC) return;
    ReceivedPkg receivedPkg;
    uint64_t identifier = (uint64_t)pkg.srcMacAddr[0] | ((uint64_t)pkg.srcMacAddr[1] << 8) |
((uint64_t)pkg.srcMacAddr[2] << 16) | ((uint64_t)pkg.srcMacAddr[3] << 24) |
((uint64_t)pkg.srcMacAddr[4] << 32) | ((uint64_t)pkg.srcMacAddr[5] << 40);
        if (_receivedPkgs.find(identifier) != _receivedPkgs.end()) {
        receivedPkg = _receivedPkgs.at(identifier);
        if (pkg.dataPkgId != receivedPkg.dataPkgId) {
            _receivedPkgs.erase(identifier);
        } else if ((receivedPkg.partId + 1) != pkg.partId) {
             _receivedPkgs.erase(identifier);
        } else {
            receivedPkg.partId = pkg.partId;
            receivedPkg.data.insert(receivedPkg.data.end(), pkg.payload, pkg.payload +
pkg.payloadSize);
            if (receivedPkg.partId == pkg.partsCnt) {
                _receivedPkgCallback(receivedPkg.srcMacAddr, receivedPkg.dataPkgId,
receivedPkg.data);
                _receivedPkgs.erase(identifier);
```

```
            }
            return;
        }
    }
    if (pkg.partId != 1) return;
    if (pkg.partsCnt == pkg.partId) {
        _receivedPkgCallback(std::vector<uint8_t>(pkg.srcMacAddr, pkg.srcMacAddr +
LORA_PKG_SRC_MAC_ADDR_SIZE), pkg.dataPkgId, std::vector<uint8_t>(pkg.payload, pkg.payload +
pkg.payloadSize));
    } else {
        receivedPkg.srcMacAddr = std::vector<uint8_t>(pkg.srcMacAddr, pkg.srcMacAddr +
LORA_PKG_SRC_MAC_ADDR_SIZE);
        receivedPkg.dataPkgId = pkg.dataPkgId;
        receivedPkg.partId = pkg.partId;
        receivedPkg.data.assign(pkg.payload, pkg.payload + pkg.payloadSize);
        _receivedPkgs.insert(std::pair<uint64_t, ReceivedPkg>(identifier, receivedPkg));
    }
    return;
}
```

Рис. 13. Обробка даних

Завершивши реалізацію *LoraTransport*, потрібно оновити частину коду, що ініціалізує пристрій збору вимірювань. Цей код запускає зчитування даних із багатоканальної радіотехнічної системи на основі FPGA, конвертує їх у структуру *DataPackage* та передає через *LoraTransport*. Послідовність обробки і передачі показано на рис. 14.

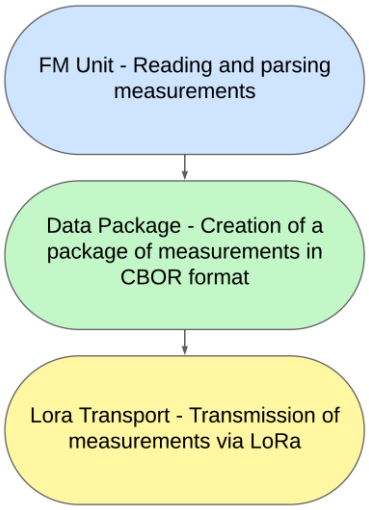

Рис. 14. Послідовність збору і передачі вимірювань

Оновлений код ініціалізації і передачі вимірювань, рис. 15, у файлі *main.cpp*.

```
void setup() {
  fm_begin();
  uint8_t mac[6];
  WiFi.macAddress(mac);
  macAddr = std::vector<uint8_t>(mac, mac + 6);
  loraTransport = new LoraTransport(macAddr, transportErrorCallback);
}

void loop() {
  if (fm_is_data_exist()) {
    FmData data;
    if (fm_get_data(data)) {
      DataPackage pkg;
      uint32_t pkgId = generate_pkg_id();
      pkg.setMacAddr(macAddr);
      pkg.setPkgId(pkgId);
      pkg.setTimeStampMs(millis());
      pkg.setFmData(data);
      loraTransport->sendBlocking(pkgId, pkg.getRawData());
    }
  }
  vTaskDelay(10);
}
```

Рис. 15. Послідовність збору і передачі вимірювань

На цьому етапі пристрій збору вимірювань повністю реалізований. Наступний крок — створення прототипу концентратора. Концентратор складатиметься з трьох основних блоків і драйверів для

вихідних інтерфейсів (рис. 16). Деякі блоки, як-от *LoraTransport* і *DataPackage*, вже реалізовані для пристрою збору вимірювань, тому їх повторна розробка не потрібна. Інші компоненти, такі як *PackageManager* і драйвери для USB, Wi-Fi та BLE, виходять за межі цієї статті.

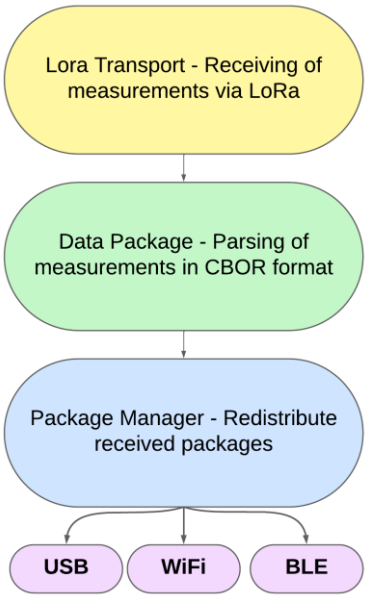

Рис. 16. Послідовність обробки вимірювань на стороні концентратора

Нижче наведено приклад реалізації концентратора, який забезпечує отримання вимірювань від пристроїв збору, їх обробку та виведення інформації на UART.

Ініціалізація концентратора, переведення його у режим приймача і реєстрація зворотного виклику для знайдених пакетів, рис. 17.

```
uint8_t mac[6];
WiFi.macAddress(mac);
macAddr = std::vector<uint8_t>(mac, mac + 6);
loraTransport = new LoraTransport(macAddr, transportErrorCallback);
LOGI(TAG, "LoraTransport initialized");
loraTransport->startReceiving(receivedPkgCallback);
```

Рис. 17. Ініціалізація концентратора

Пошук пакетів від пристроїв збору, рис 18, та їх обробка, рис. 19.

.

```
loraTransport->loop();
```

Рис. 18. Пошук пакетів

```
void receivedPkgCallback(const std::vector<uint8_t> &srcMacAddr, uint32_t pkgId, const
std::vector<uint8_t> &data) {
  DataPackage dataPackage;
  dataPackage.setRawData(data);
  if (dataPackage.isMacAddrExist()) {
    LOGI(TAG, "MAC address: %02X:%02X:%02X:%02X:%02X:%02X", dataPackage.getMacAddr()[0],
dataPackage.getMacAddr()[1], dataPackage.getMacAddr()[2], dataPackage.getMacAddr()[3],
dataPackage.getMacAddr()[4], dataPackage.getMacAddr()[5]);
  }
  if (dataPackage.isPkgIdExist()) {
    LOGI(TAG, "Package ID: %d", dataPackage.getPkgId());
  }
  if (dataPackage.isTimeStampExist()) {
    LOGI(TAG, "Time stamp: %d", dataPackage.getTimeStampMs());
  }
  if (dataPackage.isFmDataExist()) {
    FmData fmData = dataPackage.getFmData();
    LOGI(TAG, "Fm data freqs: %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d", fmData.freqs[0],
fmData.freqs[1], fmData.freqs[2], fmData.freqs[3], fmData.freqs[4], fmData.freqs[5],
fmData.freqs[6], fmData.freqs[7], fmData.freqs[8], fmData.freqs[9], fmData.freqs[10],
fmData.freqs[11]);
    LOGI(TAG, "Fm data: temp: %d, accel_x: %d, accel_y: %d, accel_z: %d, gyro_x: %d, gyro_y:
%d, gyro_z: %d", fmData.temp, fmData.accel_x, fmData.accel_y, fmData.accel_z, fmData.gyro_x,
fmData.gyro_y, fmData.gyro_z);
  }
}
```

Рис. 19. Обробка пакетів

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

У роботі запропоновано підхід до створення багатоканальної вимірювальної системи на основі FPGA із використанням модуля Lilygo LORA32, що дозволило ефективно поєднати переваги високопродуктивних обчислювальних систем із можливостями бездротового передавання даних. Основною метою було забезпечення надійного збору, обробки та передачі вимірювань у системах із частотними сенсорами фізичних величин, що досягнуто завдяки розробленому протоколу передачі даних у форматі CBOR та інтеграції технології LoRa. Результати дослідження продемонстрували такі ключові переваги розробленої системи:

1. Гнучкість і масштабованість: Використання модуля Lilygo LORA32 із підтримкою LoRa, WiFi та Bluetooth забезпечує можливість адаптації системи до різних сценаріїв використання, включаючи вимірювання у віддалених або важкодоступних місцях.
2. Ефективність обробки даних: Формат CBOR дозволив мінімізувати обсяг даних для передачі, що знижує навантаження на канали зв'язку та підвищує швидкість передачі.
3. Енергоефективність: Завдяки використанню LoRa протоколу досягнуто значної економії енергії, що дозволяє використовувати систему в умовах з обмеженими ресурсами живлення.
4. Універсальність: Інтеграція FPGA з LoRa модулем створює платформу для подальшого розширення функціональності, зокрема підтримки нових типів сенсорів і способів обробки даних.

Розробка системи стала важливим кроком у напрямку створення сучасних вимірювальних систем із високою точністю, надійністю та можливістю бездротової передачі. Подальший розвиток цієї платформи включає впровадження нових методів обробки даних, підвищення продуктивності концентратора та створення інтегрованих рішень для моніторингу в реальному часі. Це відкриває перспективи для широкого використання системи у сфері IoT, автоматизації виробництва, моніторингу довкілля та багатьох інших галузях.

Література

1. Кофанов, В. Л., Осадчук, О. В., & Гаврілов, Д. В. (2006). *Лабораторний практикум з дослідження цифрових пристроїв на основі САПР MAX+PLUS II*. Вінниця: УНІВЕРСУМ-Вінниця.
2. Осадчук, О. В., Осадчук, Я. О., & Скощук, В. К. (2021). Багатоканальний частотомір на програмованій логічній інтегральній схемі для радіовимірювальної системи з частотними сенсорами фізичних величин. *Вісник Хмельницького національного університету, (6)(303),* 186–194. https://doi.org/10.31891/2307-5732-2021-303-6-186-194
3. Осадчук, О. В., Осадчук, Я. О., & Скощук, В. К. (2023). Використання ядра NIOS II у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин. *Measuring and Computing Devices in Technological Processes, (1),* 137–148. https://doi.org/10.31891/2219-9365-2023-73-1-19
4. Осадчук, О. В., Осадчук, Я. О., & Скощук, В. К. (2023). Удосконалення багатоканальної радіотехнічної системи на FPGA для частотних перетворювачів фізичних величин з підтримкою цифрових сенсорів. *Measuring and Computing Devices in Technological Processes, (2),* 72–82. https://doi.org/10.31891/2219-9365-2023-74-10
5. Осадчук, О. В., Осадчук, Я. О., & Скощук, В. К. (2024). Інтеграція Lilygo Lora32 у багатоканальну радіотехнічну систему на FPGA для частотних перетворювачів фізичних величин. *Measuring and Computing Devices in Technological Processes, (4),* 74–83. https://doi.org/10.31891/2219-9365-2024-80-10
6. Altera. (2016). *Nios II Processor Reference Handbook.* San Jose: Altera.
7. Borgonovo, D., Heldwein, M. L., & Mussa, S. A. (2008). Application of the NIOS II processor-FPGA on the digital control of a single-phase PFC rectifier. *2008 11th Workshop on Control and Modeling for Power Electronics (COMPEL),* Zurich, Switzerland, 1–7. https://doi.org/10.1109/COMPEL.2008.4634702
8. Safarpour, M., Hautala, I., & Silvén, O. (2018). An embedded programmable processor for compressive sensing applications. *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC),* Tallinn, Estonia, 1–5. https://doi.org/10.1109/NORCHIP.2018.8573494
9. Tayara, H., Ham, W., & Chong, K. T. (2016). A real-time marker-based visual sensor based on an FPGA and a soft core processor. *Sensors, 16*(12), 2139. https://doi.org/10.3390/s16122139
10. Magdaleno, E., Rodríguez, M., Pérez, F., Hernández, D., & García, E. (2014). A FPGA embedded web server for remote monitoring and control of smart sensors networks. *Sensors, 14*(1), 416–430. https://doi.org/10.3390/s140100416
11. Lilygo. (n.d.). *LoRa32 V2.1_1.6 – LILYGO.* Retrieved from https://www.lilygo.cc
12. CBOR. (n.d.). *CBOR – Concise Binary Object Representation.* Retrieved from https://cbor.io/
13. RadioLib. (n.d.). *RadioLib – Arduino Library for Wireless Communication Modules.* Retrieved from https://github.com/jgromes/RadioLib

References

1. Kofanov, V. L., Osadchuk, O. V., & Havrilov, D. V. (2006). Laboratornyi praktykum z doslidzhennia tsyfrovykh prystroiv na osnovi SAPR MAX+PLUS II. Vinnytsia: UNIVERSUM-Vinnytsia.
2. Osadchuk, O. V., Osadchuk, Ya. O., & Skoshchuk, V. K. (2021). Bahatokanalnyi chastotomir na prohramovanii lohichnii intehralnii skhemi dlia radiovymiriuvalnoi systemy z chastotnymy sensoramy fizychnykh velychyn. Visnyk Khmelnytskoho natsionalnoho universytetu, (6)(303), 186–194. https://doi.org/10.31891/2307-5732-2021-303-6-186-194
3. Osadchuk, O. V., Osadchuk, Ya. O., & Skoshchuk, V. K. (2023). Vykorystannia yadra NIOS II u bahatokanalnomu chastotomiri na FPGA dlia radiotekhnichnoi systemy z chastotnymy sensoramy fizychnykh velychyn. Measuring and Computing Devices in Technological Processes, (1), 137–148. https://doi.org/10.31891/2219-9365-2023-73-1-19
4. Osadchuk, O. V., Osadchuk, Ya. O., & Skoshchuk, V. K. (2023). Udoskonalennia bahatokanalnoi radiotekhnichnoi systemy na FPGA dlia chastotnykh peretvoriuvachiv fizychnykh velychyn z pidtrymkoiu tsyfrovykh sensoriv. Measuring and Computing Devices in Technological Processes, (2), 72–82. https://doi.org/10.31891/2219-9365-2023-74-10
5. Osadchuk, O. V., Osadchuk, Ya. O., & Skoshchuk, V. K. (2024). Intehratsiia Lilygo Lora32 u bahatokanalnu radiotekhnichnu systemu na FPGA dlia chastotnykh peretvoriuvachiv fizychnykh velychyn. Measuring and Computing Devices in Technological Processes, (4), 74–83. https://doi.org/10.31891/2219-9365-2024-80-10
6. Altera. (2016). Nios II Processor Reference Handbook. San Jose: Altera.
7. Borgonovo, D., Heldwein, M. L., & Mussa, S. A. (2008). Application of the NIOS II processor-FPGA on the digital control of a single-phase PFC rectifier. 2008 11th Workshop on Control and Modeling for Power Electronics (COMPEL), Zurich, Switzerland, 1–7. https://doi.org/10.1109/COMPEL.2008.4634702
8. Safarpour, M., Hautala, I., & Silvén, O. (2018). An embedded programmable processor for compressive sensing applications. 2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), Tallinn, Estonia, 1–5. https://doi.org/10.1109/NORCHIP.2018.8573494
9. Tayara, H., Ham, W., & Chong, K. T. (2016). A real-time marker-based visual sensor based on an FPGA and a soft core processor. Sensors, 16(12), 2139. https://doi.org/10.3390/s16122139
10. Magdaleno, E., Rodríguez, M., Pérez, F., Hernández, D., & García, E. (2014). A FPGA embedded web server for remote monitoring and control of smart sensors networks. Sensors, 14(1), 416–430. https://doi.org/10.3390/s140100416
11. Lilygo. (n.d.). LoRa32 V2.1_1.6 – LILYGO. Retrieved from https://www.lilygo.cc
12. CBOR. (n.d.). CBOR – Concise Binary Object Representation. Retrieved from https://cbor.io/
13. RadioLib. (n.d.). RadioLib – Arduino Library for Wireless Communication Modules. Retrieved from https://github.com/jgromes/RadioLib