https://doi.org/10.31891/2219-9365-2023-74-10
УДК 621.317

ОСАДЧУК Олександр
Вінницький національний технічний університет
https://orcid.org/0000-0001-6662-9141
e-mail: osadchuk.av69@gmail.com
ОСАДЧУК Ярослав
Вінницький національний технічний університет
https://orcid.org/0000-0002-5472-0797
e-mail: osadchuk.av69@gmail.com
СКОЩУК Валентин
Вінницький національний технічний університет
e-mail: skoschuk999@gmail.com

УДОСКОНАЛЕННЯ БАГАТОКАНАЛЬНОЇ РАДІОТЕХНІЧНОЇ СИСТЕМІ НА FPGA ДЛЯ ЧАСТОТНИХ ПЕРЕТВОРЮВАЧІВ ФІЗИЧНИХ ВЕЛИЧИН ПІДТРИМКОЮ ЦИФРОВИХ СЕНСОРІВ

У роботі представлено розробку функціонального розширення можливостей багатоканальної радіотехнічної системи паралельного вимірювання частотних інформативних сигналів на основі FPGA фірми Altera Cyclone IV. Основною задачею розробленої системи є вимірювання інформативного параметру сенсорів фізичних величин з частотним виходом, за допомогою підтримки цифрових сенсорів. Удосконалено багатоканальний універсальний вимірювальний прилад на основі FPGA, який має 12 паралельних вимірювальних каналів для сенсорів з частотним виходом та інтегрованим мікропроцесорним ядром NIOS II. Удосконалено внутрішню конфігурацію мікропроцесорного ядра NIOS II, синтезовано блок ядра, створено апаратну реалізацію I2C інтерфейсу, додано програмну підтримку I2C протоколу, реалізовано програмну підтримку трьох цифрових сенсорів, оновлено і повторно синтезовано загальну схему. Інтеграція I2C протоколу у багатоканальний частотомір дозволяє одночасне вимірювання значень з двох типів сенсорів, що значно розширює спектр можливих шляхів використання радіотехнічної інформаційно-вимірювальної системи.

Ключові слова: I2C, NIOS II, FPGA, багатоканальний частотомір, сенсор з частотним виходом, радіовимірювальний перетворювач фізичних величин, частота.

OSADCHUK Oleksandr, OSADCHUK Iaroslav, SKOSHCHUK Valentyn
Vinnytsia National Technical University

IMPROVEMENT OF MULTI-CHANNEL RADIO ENGINEERING SYSTEM ON FPGA FOR FREQUENCY CONVERTERS OF PHYSICAL QUANTITIES WITH THE SUPPORT OF DIGITAL SENSORS

The work presents the development of the functional expansion of the multi-channel radio technical system of parallel measurement of frequency informative signals based on the Altera Cyclone IV FPGA. The main task of the developed system is to measure the informative parameter of sensors of physical quantities with frequency output, using the support of digital sensors. Improved multi-channel universal measuring device based on FPGA, which has 12 measurement channels for sensors with frequency output and integrated NIOS II microprocessor core. The internal configuration of the NIOS II microprocessor core was reworked, the core block was re-synthesized, the I2C hardware implementation was created, software support for the I2C protocol was added, software support for three digital sensors was implemented, the general circuit was updated and re-synthesized. Integration of the I2C protocol into a multi-channel frequency meter will allow simultaneous measurement of values from two types of sensors, which will significantly expand the range of possible ways of using the system. One of the most common protocols - I2C was chosen for communication with digital sensors. This bus is one of the modifications of serial data exchange protocols. In the standard mode, the transmission of serial 8-bit data is provided at a speed of up to 100 kbit/s, and up to 400 kbit/s in the "fast" mode. In the previous version of the scheme, the NIOS II core was already integrated and all the main blocks were redesigned for the core interfaces. Therefore, to integrate the support of digital sensors into the existing scheme, it is only necessary to create a kernel interface for I2C, implement the I2C protocol in the form of a hardware block, and write software for I2C support. As a result, the scheme for the device was updated, one new block was added to it, which describes the I2C hardware part and the interface for the core. The maximum allowable number of chips connected to one I2C bus is limited by the maximum capacity of the bus, which is 400 pF. The UART digital protocol is used as the output interface. Also, to support the I2C protocol and digital sensors, software was developed that complements the previous version of the implementation and allows processing data from frequency meters and digital sensors at the same time.

Keywords: FPGA, NIOS II, I2C, sensor with frequency output, multichannel frequency meter, frequency, radio measuring transducers of physical quantities.

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

На даний час важко знайти широко доступні рішення які б задовольнили вимоги, щодо одночасного вимірювання значень сенсорів з цифровими і частотними виходами. Сенсори із цифровими виходами більш

поширені ніж із частотними, також різновидів цифрових сенсорів існує набагато більше ніж частотних. Можливість використання у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин [1-2], одночасно двох типів сенсорів, значно розширить спектр її можливих шляхів використання. Підтримка ядра NIOS II [3-7] дозволить одночасно аналізувати значення із двох типів сенсорів і застосовувати до них різні алгоритми фільтрації і обробки.

Аналіз останніх досліджень

У роботі [2] авторами реалізовано багатоканальну систему паралельного вимірювання частоти на основі FPGA фірми Altera Cyclone IV EP4CE10F17C8 і мікропроцесорного ядра NIOS II, основною задачею якої є вимірювання інформативного параметру сенсорів фізичних величин з частотним виходом. Розроблена система удосконалює багатоканальний універсальний вимірювальний прилад на основі FPGA, який має 12 паралельних вимірювальних каналів для сенсорів з частотними виходами [1], рис. 1. Для цього автори переробили блоки: лічильників імпульсів, блок обробки даних, UART передавач, згенерували і налаштували мікропроцесорне ядро Nios II, повторно синтезували схему, рис. 2.

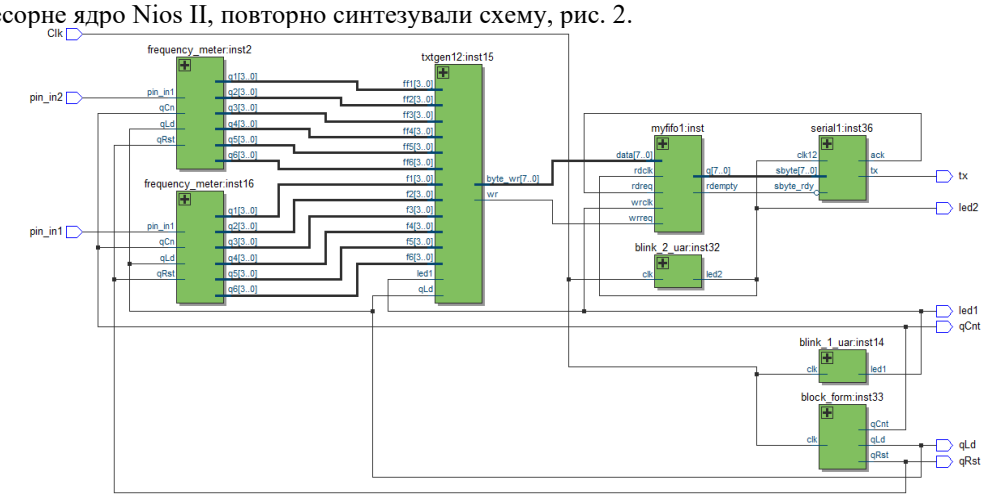

Рис. 1. Схема багатоканального частотоміра

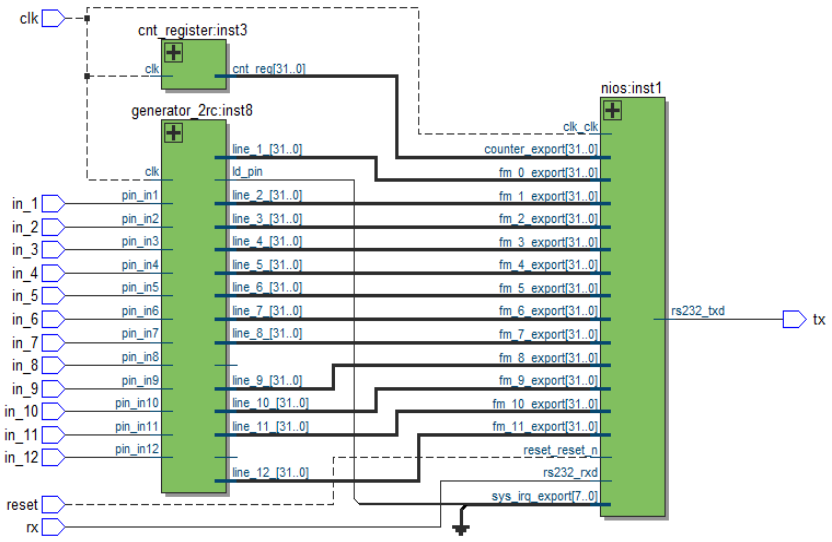

Рис. 2. Схема багатоканального частотоміра з використанням ядра NIOS II

Інтеграція NIOS II у багатоканальний частотомір дозволила зробити систему гнучкою, додати попередню обробку, фільтрацію отриманих даних і змінювати кількість частотомірів, без зміни алгоритму обробки даних, що являлося неможливим у попередній реалізації [1]. Для створення проекту під FPGA використовувалася версію Quartus 15.1 [8].

Для інтеграція мікропроцесорної системи в існуючу схему частотоміра рис. 1, першим етапом було генерування і налаштування мікропроцесорної системи Nios II за допомогою утиліти Qsys [9]. Після чого блок лічильника імпульсів був перероблений, під інтерфейс ядра. Такі блоки як: обробка даних з лічильників, UART передавач, повністю втратили свою актуальність, їхній функціонал перенесли на сторно ядра.

Для перевірки драйвера частотомірів до системи було під'єднано генератор частоти на 85 кГц до першого каналу. Після зчитування значень частотомірів відбувається формування повідомлення, кожне значення розділене символом «\t». Сформоване повідомлення відправляється до USB порту ПК через конвертор PL2303, рис. 3.

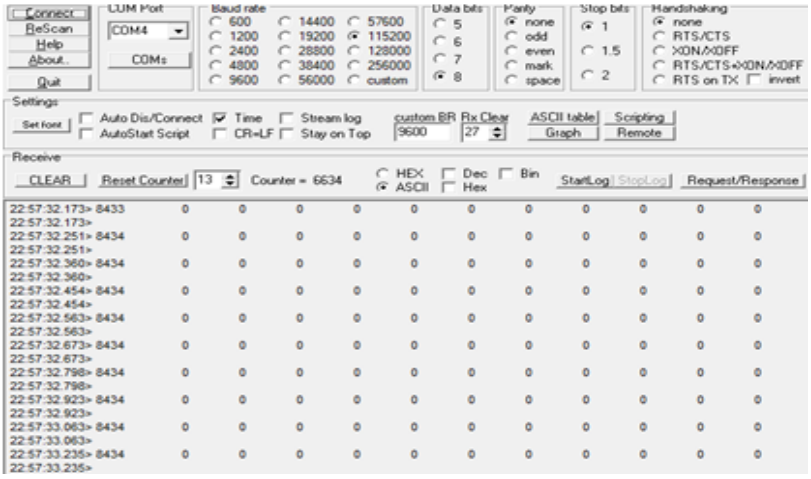

Рис. 3. Результат роботи процесорної системи з під'єднаними до неї частотними датчиками

Формулювання цілей статті

Метою роботи є: інтеграція I2C протоколу у багатоканальний частотомір на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин [1-2], що дозволить розширить спектр її можливих шляхів використання.

Теоретичні та експериментальні дослідження

Для комунікації із цифровими сенсорами було обрано один із найпоширеніших протоколі - I2C [10]. Ця шина є однією з модифікацій послідовних протоколів обміну даних. У стандартному режимі забезпечується передача послідовних 8-бітних даних зі швидкістю до 100 кбіт/с, і до 400 кбіт/с в "швидкому" режимі. Для здійснення процесу обміну інформацією по I2C шині, використовується всього два сигнали лінія даних SDA лінія синхронізації SCL, рис. 4. Проста двохпровідна послідовна шина I2C мінімізує кількість з'єднань між IC, що призводить до зменшення об'єму комунікаційних сполучень. Як результат - друковані плати стають простішими і технологічними при виготовленні.

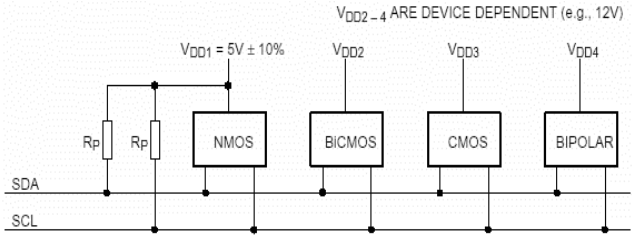

Рис. 4. Підключення декількох пристроїв до спільної шини

У попередній версії схеми [2], рис. 2, вже було інтегровано ядро NIOS II і перероблено всі основні блоки під інтерфейси ядра. Для інтеграції підтримки цифрових сенсорів у існуючу схему необхідно виконати наступне: створити інтерфейс ядра для підтримки I2C блоку, реалізувати I2C протокол у вигляді окремого блоку для взаємодії із цифровими сенсорами, написати ПЗ для підтримки I2C у ядрі, створити драйвери для підтримки цифрових сенсорів.

Першим етапом являється створення інтерфейсу у ядрі під I2C блок. Генерація і налаштування мікропроцесорної системи відбувається за допомогою утиліти Qsys [9]. «I2C» - інтерфейс для підключення блоку, рис. 5, інтерфейс розділений на дві частини: 10 розрядна вихідна шина з підтримкою переривань (на стороні процесора); 17 розрядна вхідна шина, рис. 6.

Рис. 5. Графічне відображення інтерфейсу для блоку «I2C»

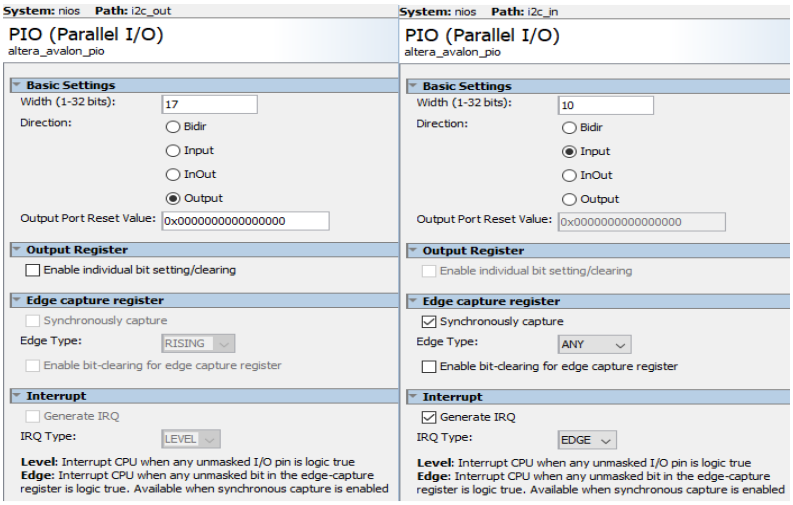

Рис. 6. Параметри налаштування інтерфейсу для блоку «I2C»

Другим етапом являється реалізація апаратної частини I2C протоколу. I2C шина використовується для взаємодії із цифровими сенсорами. Реалізація даного протоколу відсутня у стандартній бібліотеці, тому, він був реалізований апаратно у вигляді окремого системного блоку, рис. 7. На рис. 8 зображено процес передачі даних I2C компонентом.

Призначення входів/виходів:

1. clk – сигнал від зовнішнього тактового генератора;
2. reset_n – очищення внутрішнього стану до початкових значень;
3. ena – керує дозволом на виконання транзакцій;
4. addr – 7-ми розрядна шина значення якої відповідають адресу веденого пристрою;
5. rw – вказує напрямок передачі даних;
6. data_wr – 8-ми розрядна шина значення якої відповідає байту даних призначеного для відправки веденому;
7. busy – вказує на процес виконання транзакції;
8. data_rd – 8-ми розрядна шина,яка зберігає зчитаний байт з веденого;
9. ack_error – вказує на виникнення помилки під час транзакції;
10. sda – послідовна лінія даних;
11. scl – послідовна лінія тактування.

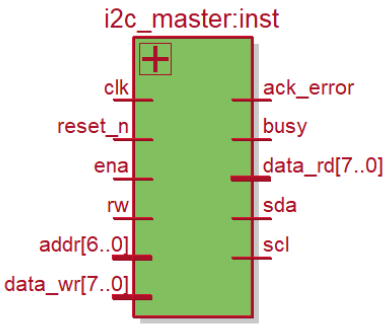

Рис. 7. Блок який реалізує апаратну частину I2C приймача/передавача

ISSN 2219-9365

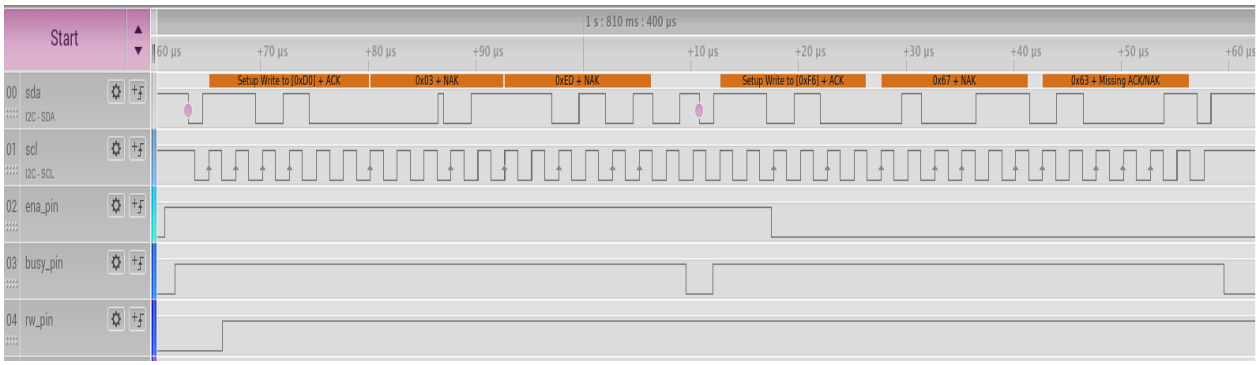

Рис. 8. Відображення сигналів на усіх лініях інтерфейсу I2C шини при спробі передачі даних

Третім етапом являється реалізація програмної частини I2C протоколу. Для взаємодії з I2C компонентом було написано драйвер який розділений на дві частини:

1) Нижній рівень – наближений до апаратного рівня, інкапсулює роботу з апаратними інтерфейсами;

2) Верхній рівень - наближений до програмного забезпечення користувача, використовує незмінний інтерфейс який реалізується нижнім рівнем, для побудови високорівневих політик протоколу I2C.

Основними програмними компонентами нижнього рівня являються дві функції які використовують регістри процесорної системи створені на першому етапі. Завдяки ним відбувається переключення між різними станами I2C блоку, запис/зчитування даних, отримання інформації про стан I2C шини, рис. 9-10.

```
alt_8 i2c_hal_reg_write(alt_u8 reg_mask, alt_u8 reg_offset, alt_u8 value) {
    // Read last reg value and clear bits for set new value
    alt_u32 reg_value = IORD_ALTERA_AVALON_PIO_DATA(I2C_OUT_BASE) & (~((alt_u32)reg_mask << reg_offset));
    // Update value for reg
    reg_value |= ((alt_u32)value & reg_mask) << reg_offset;
    // Write new value into register
    IOWR_ALTERA_AVALON_PIO_DATA(I2C_OUT_BASE, reg_value);
    return RES_OK;
}
```

Рис. 9. Реалізація функції запису у регістри I2C компонента

```
alt_8 i2c_hal_reg_read(alt_u8 reg_mask, alt_u8 reg_offset, alt_u8 *value) {
    // check if pointer is valid
    if (!value)
        return RES_ERR;
    // cut part of common register value
    *value = (IORD_ALTERA_AVALON_PIO_DATA(I2C_IN_BASE) & ((alt_u32)reg_mask << reg_offset)) >> reg_offset;

    return RES_OK;
}
```

Рис. 10. Реалізація функції зчитування з регістрів I2C компонента

На верхньому рівні знаходяться реалізації функцій які використовуючи реалізації нижнього рівня, описують алгоритми запису і зчитування регістрів з підключених до шини пристроїв, рис. 11-12.

```
alt_8 i2c_write_bytes(alt_u8 dev_addr, alt_u8 reg_addr, alt_u8 *w_data_arr, alt_u8 len) {
    alt_u8 i = 0x00;
    alt_u32 irq_state = 0x00;

    if (i2c_is_busy() || !w_data_arr || !len)
        return RES_ERR;

    // Disable all interrupts
    irq_state = sys_irq_critical_section_begin();

    i2c_hal_set_slave_addr(dev_addr);
    i2c_hal_set_rw(I2C_WRITE);
    i2c_hal_set_write_data(reg_addr);
    i2c_hal_set_ena(I2C_ENABLE_TRANSMISSION);

    // Wait stage of set register value
    _i2c_wait_for_busy(TRUE);

    for (i = 0x00; i < len; i++) {
        i2c_hal_set_write_data(w_data_arr[i]);
        if (i2c_is_err()) {
            i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
            sys_irq_critical_section_end(irq_state);
            sys_printf("\r\n%s() -> Err at set register value stage, dev: %d, reg: %d\r\n",
                       __func__, dev_addr, reg_addr + i);
            return RES_ERR;
        }
        // Wait for write register value stage
        _i2c_wait_for_busy(FALSE);
        _i2c_wait_for_busy(TRUE);
    }

    i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
    // End of transmission
    _i2c_wait_for_busy(FALSE);
    if (i2c_is_err()) {
        i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
        sys_irq_critical_section_end(irq_state);
        sys_printf("\r\n%s() -> Err at end of read process, dev: %d, reg: %d\r\n",
                   __func__, dev_addr, reg_addr + i);
        return RES_ERR;
    }

    // Enable interrupts
    sys_irq_critical_section_end(irq_state);

    return RES_OK;
```

Рис. 11. Реалізація алгоритму запису байтів у ведений пристрій

```
alt_8 i2c_read_bytes(alt_u8 dev_addr, alt_u8 reg_addr, alt_u8 *r_data_arr, alt_u8 len) {
    alt_u8 i = 0x00;
    alt_u32 irq_state = 0x00;

    if (i2c_is_busy() || !r_data_arr || !len)
        return RES_ERR;

    // Disable all interrupts
    irq_state = sys_irq_critical_section_begin();

    // Set up I2C registers for read
    i2c_hal_set_slave_addr(dev_addr);
    i2c_hal_set_rw(I2C_WRITE);
    i2c_hal_set_write_data(reg_addr);
    i2c_hal_set_ena(I2C_ENABLE_TRANSMISSION);

    // Wait for start transmission
    _i2c_wait_for_busy(TRUE);
    i2c_hal_set_rw(I2C_READ);
    if (i2c_is_err()) {
        i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
        sys_irq_critical_section_end(irq_state);
        sys_printf("\r\n%s() -> Err at start transmission, dev: %d, reg: %d\r\n",
                __func__, dev_addr, reg_addr);
        return RES_ERR;
    }

    // Wait read stage
    _i2c_wait_for_busy(FALSE);
    for (i = 0x00; i < len; i++) {
        _i2c_wait_for_busy(TRUE);
        // Disable transmission before end of current
        // transaction for avoid starting new transaction
        if (i == len - 1)
            i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);

        _i2c_wait_for_busy(FALSE);
        i2c_hal_get_read_data(r_data_arr + i);
        if (i2c_is_err()) {
            i2c_hal_set_ena(I2C_DISABLE_TRANSMISSION);
            sys_irq_critical_section_end(irq_state);
            sys_printf("\r\n%s() -> Err during read register, dev: %d, reg: %d\r\n",
                    __func__, dev_addr, reg_addr + i);
            return RES_ERR;
        }
    }

    // Enable interrupts
    sys_irq_critical_section_end(irq_state);

    return RES_OK;
}
```

Рис. 12. Реалізація алгоритму зчитування байтів з веденого пристрою

Для перевірки I2C драйвера до системи було під'єднано модуль MPU6050 який містить три цифрових датчика (гіроскоп, акселерометр, датчик температури), взаємодія з ними відбувається за допомогою I2C-шини.

Для отримання даних з трьох датчиків було реалізовано драйвер MPU6050, він використовує I2C драйвер і містить три основних функції які використовують програмний інтерфейс I2C протоколу для обміну даними з датчиками, рис. 13-15.

```
alt_8 mpu6050_accel(mpu6050_t *mpu_dev, alt_16 *x, alt_16 *y, alt_16 *z) {
    alt_u8 accel_data[MPU6050_ACCEL_REG_NUM] = {0x00};

    if (!mpu_dev || !_mpu6050_get_dev_by(mpu_dev->addr) || !x || !y || !z) {
        sys_printf("\r\n%s() -> Error incorrect parameters\r\n", __func__);
        return RES_ERR;
    }

    if (i2c_read_bytes(mpu_dev->addr, MPU6050_RA_ACCEL_XOUT_H, accel_data, MPU6050_ACCEL_REG_NUM)) {
        sys_printf("\r\n%s() -> Error read from dev: %d, reg: %d\r\n", __func__, mpu_dev->addr, MPU6050_RA_ACCEL_XOUT_H);
        return RES_ERR;
    }

    // Converting received data into accel x, y, z
    *x = (((alt_16)accel_data[0]) << 8) | accel_data[1];
    *y = (((alt_16)accel_data[2]) << 8) | accel_data[3];
    *z = (((alt_16)accel_data[4]) << 8) | accel_data[5];

    *x = *x < 0? (alt_16)(mpu_dev->acc_coeff * (*x * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *x);
    *y = *y < 0? (alt_16)(mpu_dev->acc_coeff * (*y * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *y);
    *z = *z < 0? (alt_16)(mpu_dev->acc_coeff * (*z * -1)) * -1 : (alt_16)(mpu_dev->acc_coeff * *z);

    return RES_OK;
}
```

Рис. 13. Зчитування значень з акселерометра

```
alt_8 mpu6050_gyro(mpu6050_t *mpu_dev, alt_16 *x, alt_16 *y, alt_16 *z) {
    alt_u8 gyro_data[MPU6050_GYRO_REG_NUM] = {0x00};

    if (!mpu_dev || !_mpu6050_get_dev_by(mpu_dev->addr) || !x || !y || !z) {
        sys_printf("\r\n%s() -> Error incorrect parameters\r\n", __func__);
        return RES_ERR;
    }

    if (i2c_read_bytes(mpu_dev->addr, MPU6050_RA_GYRO_XOUT_H, gyro_data, MPU6050_GYRO_REG_NUM)) {
        sys_printf("\r\n%s() -> Error read from dev: %d, reg: %d\r\n", __func__, mpu_dev->addr, MPU6050_RA_GYRO_XOUT_H);
        return RES_ERR;
    }

    // Converting received data into gyro x, y, z
    *x = (((alt_16)gyro_data[0]) << 8) | gyro_data[1];
    *y = (((alt_16)gyro_data[2]) << 8) | gyro_data[3];
    *z = (((alt_16)gyro_data[4]) << 8) | gyro_data[5];

    *x = *x < 0? (alt_16)(mpu_dev->gyro_coeff * (*x * -1)) * -1 : (alt_16)(mpu_dev->gyro_coeff * *x);
    *y = *y < 0? (alt_16)(mpu_dev->gyro_coeff * (*y * -1)) * -1 : (alt_16)(mpu_dev->gyro_coeff * *y);
    *z = *z < 0? (alt_16)(mpu_dev->gyro_coeff * (*z * -1)) * -1 : (alt_16)(mpu_dev->gyro_coeff * *z);

    return RES_OK;
}
```

Рис. 14. Зчитування значень з гіроскопа

```
alt_8 mpu6050_temp(mpu6050_t *mpu_dev ,alt_u16 *temp) {
    alt_u8 temp_data[MPU6050_TEMP_REG_NUM] = {0x00}, guard_cnt = MPU6050_TRYS_TO_READ;

    if (!mpu_dev || !_mpu6050_get_dev_by(mpu_dev->addr) || !temp) {
        sys_printf("\r\n%s() -> Error incorrect parameters\r\n", __func__);
        return RES_ERR;
    }

    read_temp:
    if (i2c_read_bytes(mpu_dev->addr, MPU6050_RA_TEMP_OUT_H, temp_data, MPU6050_TEMP_REG_NUM)) {
        sys_printf("\r\n%s() -> Error read from dev: %d, reg: %d\r\n", __func__, mpu_dev->addr, MPU6050_RA_TEMP_OUT_H);
        return RES_ERR;
    }

    // Check if data reads correctly
    if (temp_data[0] == 0 && temp_data[1] == 0) {
        // Read allowed only guard_cnt numbers
        if (guard_cnt--) {
            goto read_temp;
        } else {
            //sys_printf("\r\n%s() -> Error max count of read temperature\r\n", __func__);
            return RES_ERR;
        }
    }

    // Converting received data into correct temperature value
    *temp = (((alt_u16)temp_data[0]) << 8) | temp_data[1];
    *temp = ((float)*temp / 340) + 36.53;

    return RES_OK;
}
```

Рис. 15. Зчитування значень з температурного датчика

Функція яка використовує реалізований драйвер спершу ініціалізує його, після чого послідовно викликає зчитування даних з акселерометра, гіроскопа і температурного датчика, реалізація функції зображена на рис. 17. Після зчитування отриманих значень відбувається формування повідомлення з ними, у послідовності їх зчитування, кожне значення розділене символом «\t». Сформоване повідомлення відправляється для передачі на UART, до якого під'єднаний конвертор PL2303 який у свою чергу під'єднується до USB порту ПК. Перші 12 значень відповідають значенням частотомірів, наступне – значення температури, наступні три – значення акселерометра і останні три – значення гіроскопа, рис. 16.

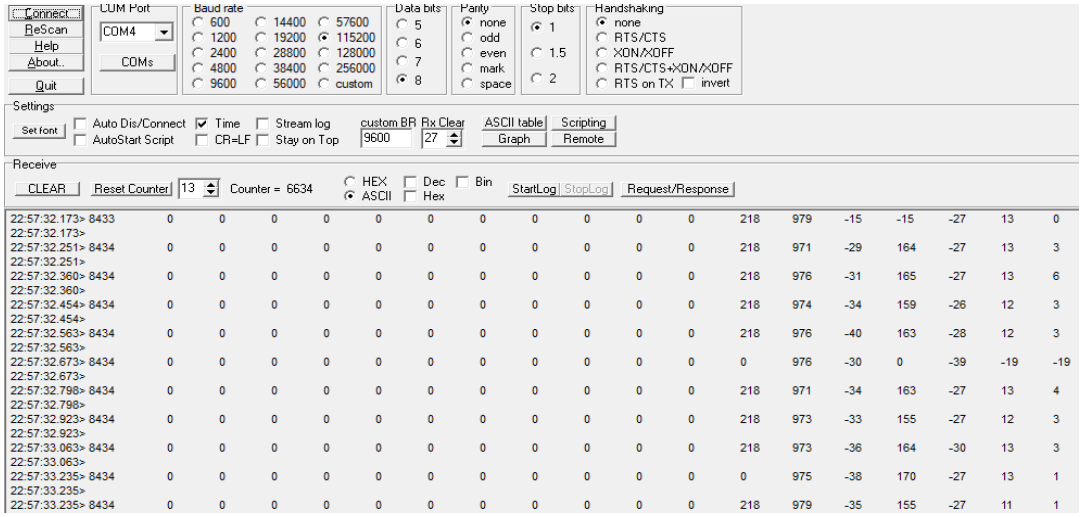

Рис. 16. Результат роботи системи з під'єднаними I2C датчиками

```
int main() {
    sys_printf("Frequency meter initializing\r\n");
    if (fm_init())
        sys_printf("Error during frequency meter initializing\r\n");

    sys_printf("I2C initializing\r\n");
    if (i2c_init())
        sys_printf("Error during I2C initializing\r\n");

    sys_printf("MPU6050 initializing\r\n");
    if (mpu6050_init(&mpu_dev))
        sys_printf("Error during MPU6050 initializing\r\n");

    /* Event loop never exits. */
    while (1) {
        alt_16 x = 0x00, y = 0x00, z = 0x00;
        alt_u16 temp = 0x00;
        alt_u8 i = 0x00;

        for (; i < CHANNEL_MAX; i++)
            data_printf(i == 0? "%d" : "\t%d", fm_get_freq(i));

        mpu6050_temp(&mpu_dev, &temp);
        data_printf("\t%d", temp);

        mpu6050_accel(&mpu_dev, &x, &y, &z);
        data_printf("\t%d\t%d\t%d", x, y, z);

        x = y = z = 0x00;
        mpu6050_gyro(&mpu_dev, &x, &y, &z);
        data_printf("\t%d\t%d\t%d\r\n", x, y, z);

        data_printf("\r\n");
        sys_delay_ms(100);
    }

    return 0;
}
```

Рис. 17. Зчитування значень з драйвера MPU6050 і вивід отриманих даних на UART

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Синтезовано схему для багатоканальної системи паралельного вимірювання частоти на основі FPGA фірми Altera Cyclone IV, яка має 12 паралельних вимірювальних каналів для сенсорів з частотним виходом, базується на мікропроцесорному ядрі NIOS II і може взаємодіяти з цифровими сенсорами фізичних величин. Для комунікації з цифровими сенсорами було обрано один із найпоширеніших протоколі - I2C. Ця шина є однією з модифікацій послідовних протоколів обміну даних. У попередній версії схеми [2], вже було інтегровано ядро NIOS II і перероблено всі основні блоки під інтерфейси ядра. Тому для інтеграції підтримки цифрових сенсорів у існуючу схему довелося лише створити інтерфейс ядра для I2C, реалізувати I2C протокол у вигляді апаратного блоку, написати ПЗ для підтримки I2C шини. В кінцевому результаті було оновлено схему для приладу [2], до неї було додано один новий блок який описує апаратну частину I2C і інтерфейс для ядра. Максимальна допустима кількість мікросхем, приєднаних до однієї I2C шини, обмежується максимальною ємністю шини яка становить 400 пФ. У якості вихідного інтерфейсу використовується цифровий протокол UART. Також для підтримки I2C протоколу і цифрових сенсорів було розроблено програмне забезпечення, яке доповнює попередню версію реалізації [2] і дозволяє обробляти дані з частотомірів і цифрових сенсорів одночасно.

Література

1. Осадчук О.В. Багатоканальний частотомір на програмованій логічній інтегральній схемі для радіовимірювальної системи з частотними сенсорами фізичних величин / Осадчук О.В., Осадчук Я.О., Скощук В.К. // Вісник Хмельницького національного університету, №6, 2021 (303) – С.186-194. DOI 10.31891/2307-5732-2021-303-6-186-194

2. Осадчук О.В. Використання ядра NIOS II у багатоканальному частотомірі на FPGA для радіотехнічної системи з частотними сенсорами фізичних величин / Осадчук О.В., Осадчук Я.О., Скощук В.К. // Вимірювальна та обчислювальна техніка в технологічних процесах, №1, 2023. –С.137-148. https://doi.org/10.31891/2219-9365-2023-73-1-19

3. Nios II Processor Reference Handbook. – San Jose: Altera, 2016. – 260 с.

4. Borgonovo D. Application of the NIOS II processor-FPGA on the digital control of a single-phase PFC rectifier," / D. Borgonovo, M. L. Heldwein and S. A. Mussa // 2008 11th Workshop on Control and Modeling for Power Electronics, Zurich, Switzerland, 2008, pp. 1-7, doi: 10.1109/COMPEL.2008.4634702.

5. Safarpour M. An Embedded Programmable Processor for Compressive Sensing Applications," / M. Safarpour, I. Hautala and O. Silvén // 2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), Tallinn, Estonia, 2018, pp. 1-5, doi: 10.1109/NORCHIP.2018.8573494.
6. Tayara H. A. Real-Time Marker-Based Visual Sensor Based on a FPGA and a Soft Core Processor / Tayara H., Ham W., Chong K.T. // Sensors. 2016; 16(12):2139. https://doi.org/10.3390/s16122139
7. Magdaleno E. A FPGA Embedded Web Server for Remote Monitoring and Control of Smart Sensors Networks / Magdaleno E., Rodríguez M., Pérez F., Hernández D., García E. // Sensors. 2014; 14(1):416-430. https://doi.org/10.3390/s140100416
8. Quartus Prime Standard Edition. URL: https://fpgasoftware.intel.com/15.1/?edition=standard&platform=windows
9. Qsys System Design TutorialURL: https://www.intel.com/content/www/us/en/docs/programmable/683378/current/qsys-system-design-tutorial.html
10. THE I2C-BUS SPECIFICATION. – Amsterdam: Philips Semiconductors, 1998.

References

1. Osadchuk O.V. A multi-channel frequency counter on a programmable logic integrated circuit for a radio measuring system with frequency sensors of physical quantities / Osadchuk O.V., Osadchuk I.O., Skoshchuk V.K. // herald of the Khmelnytskyi National University, No. 6, 2021 (303) – P.186-194. DOI 10.31891/2307-5732-2021-303-6-186-194
2. Osadchuk O.V. The use of the NIOS II core in a multi-channel frequency meter on FPGA for a radio engineering system with frequency sensors of physical quantities / O. V. Osadchuk, I. O. Osadchuk, V. K. Skoshchuk // Measuring and computing equipment in technological processes, No. 1, 2023. – P.137-148. https://doi.org/10.31891/2219-9365-2023-73-1-19
3. Nios II Processor Reference Handbook. – San Jose: Altera, 2016. – 260 c.
4. Borgonovo D. Application of the NIOS II processor-FPGA on the digital control of a single-phase PFC rectifier," / D. Borgonovo, M. L. Heldwein and S. A. Mussa // 2008 11th Workshop on Control and Modeling for Power Electronics, Zurich, Switzerland, 2008, pp. 1-7, doi: 10.1109/COMPEL.2008.4634702.
5. Safarpour M. An Embedded Programmable Processor for Compressive Sensing Applications," / M. Safarpour, I. Hautala and O. Silvén // 2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), Tallinn, Estonia, 2018, pp. 1-5, doi: 10.1109/NORCHIP.2018.8573494.
6. Tayara H. A. Real-Time Marker-Based Visual Sensor Based on a FPGA and a Soft Core Processor / Tayara H., Ham W., Chong K.T. // Sensors. 2016; 16(12):2139. https://doi.org/10.3390/s16122139
7. Magdaleno E. A FPGA Embedded Web Server for Remote Monitoring and Control of Smart Sensors Networks / Magdaleno E., Rodríguez M., Pérez F., Hernández D., García E. // Sensors. 2014; 14(1):416-430. https://doi.org/10.3390/s140100416
8. Quartus Prime Standard Edition. URL: https://fpgasoftware.intel.com/15.1/?edition=standard&platform=windows
9. Qsys System Design TutorialURL: https://www.intel.com/content/www/us/en/docs/programmable/683378/current/qsys-system-design-tutorial.html
10. THE I2C-BUS SPECIFICATION. – Amsterdam: Philips Semiconductors, 1998.