



Mikrokontrolery ARM

Laboratorium 4

Krzysztof Pierczyk
1 maja 2020



WSTĘP

Czwarte zajęcia laboratoryjne kontynuowały tematykę komunikacji szeregowej. Tym razem zajęliśmy się protokołami *SPI* oraz *I²C*. Płytką ewaluacyjną STM32F411E-Disco posiada kilka układów scalonych wyposażonych w obydwa interfejsy, co umożliwiło zapoznanie się z nimi w praktyce. Ze względu na złożoność protokołu szyny *I²C* skorzystaliśmy ze sterowników **spi_master** oraz **i2c_master** dostępnych w bibliotece *libperiph* systemu ISIX. Struktura przygotowanych aplikacji została zaprojektowana tak, aby możliwa była łatwa wymiana sterowników na do wolne inne.

ZADANIE 1

Pierwsze zadanie polegało napisaniu interfejsu dla układu MEMS (*ang. MicroElectroMechanical System*) **L3GD20** umożliwiającego pomiar prędkości kątowej wokół trzech, prostopadłych do siebie osi. Każdy pomiar wykonywany jest z rozdzielczością 16-bitów przy czym zakres pomiaru jest konfigurowalny i wynosić on może kolejno ± 250 , ± 500 lub ± 2000 °/s. Konfiguracji dokonuje się poprzez zmianę wartości wewnętrznych rejestrów sensora. Układ został wyposażony zarówno w interfejs *SPI*, jak i *I²C*. Ze względu na fakt, że układ wykorzystywany w zadaniu 2 posiada tylko drugi interfejs, w tym przypadku wykorzystaliśmy komunikację po *SPI*.

Zgodnie z treścią zadania stworzona aplikacja powinno odczytywać z częstotliwością 4Hz aktualny pomiar czujnika dla wszystkich trzech osi, wyświetlać go na porcie szeregowym oraz wizualizować odczytany stan z wykorzystaniem diod LD3...LD6. Wizualizacja ta powinna polegać na zapalaniu diody wskazującej kierunek, w którym następuje obrót układu. Progiem aktywacji powinna być wartość bezwzględna równa 500 co, przy zakresie ustawionym na ± 2000 °/s, odpowiada szybkości około $\pm 30,52$ °/s. Aby urozmaicić zadanie, efekt wizualizacji zmodyfikowana tak, aby jasność diody nie była sterowana binarnie, a **poprzez płynnie zmienianą wartość wypełnienia sygnału PWM**, który ją kontroluje. W tym celu skonfigurowano układ TIM4 tak, aby na każdym z czterech kanałów wytwarzał falę prostokątną o częstotliwości 200Hz. Poziom wypełnienia był sterowany w zakresie 0...500 (co odpowiadało wypełnieniu 0...100%). Przedział ten został zmapowany na zakres pomiarowy 0...2000 °/s.

W celu zapewnienia prostej, przejrzystej i zwięzłej obsługi układu stworzona została klasa **L3GD20** udostępniająca następujące metody:

- **bool isConnected()** – odczytuje wewnętrzny rejestr WHO_AM_I. Jeżeli odczyt się powiódł, a wartość równa jest wartości przewidywanej oznacza to, że układ jest prawidłowo podłączony do magistrali i zwracana jest wartość **true**
- **int readControlRegister(uint8_t* dst, uint8_t s_address, uint8_t num)** – odczytuje wartość bloku rejestrów kontrolnych. Pierwszy z rejestrów wskazywany jest przez adres **s_address** natomiast liczba rejestrów do odczytania przez **num**.
- **int readMeasurementRegister(uint8_t* dst, uint8_t s_address, uint8_t num)** – odczytuje wartość bloku rejestrów zawierających pomiary. Pierwszy z rejestrów wskazywany jest przez adres **s_address** natomiast liczba rejestrów do odczytania przez **num**.
- **int writeControlRegister(uint8_t* src, uint8_t s_address, uint8_t num)** – zapisuje zbiór wartości do bloku rejestrów kontrolnych. Pierwszy z rejestrów wskazywany jest przez adres **s_address** natomiast liczba rejestrów do odczytania przez **num**.
- **float getRange()** – zwraca aktualny zakres czujnika

Metody odczytujące i zapisujące rejestry układu są jedynie wrapperami wokół generycznych metod **int readRegisters(uint8_t* dst, uint8_t s_address, uint8_t num)** oraz **int writeRegisters(uint8_t* src, uint8_t s_address, uint8_t num)**, które realizują kontrolę poprawności adresów i wartości podanych w liście argumentów. Generyczne metody są prywatne i wykorzystują bibliotekę *libperiph* w celu wykonywania blokowych transferów danych z oraz do układu żyroskopu. Sprawia to, że wymiana wewnętrznego mechanizmu realizującego transfery sprawdza się jedynie do **wymiany tych dwóch metod** (oraz implementacji konstruktora klasy, który realizowałby procedurę inicjalizacji).

Wymagania dotyczące sprawozdania przewidują umieszczenie w nim wszystkich kodów źródłowych, jednak ze względu na ich obszerność zdecydowano się umieścić jedynie elementy kluczowe, czyli metody odpowiedzialne za przeprowadzanie transferów danych. Pełen kod do zadanie zostanie dołączony do sprawozdania. Na Rysunku 1 przedstawiono informację wypisywaną przez program na port szeregowy.

```

int L3GD20::readRegisters(uint8_t* dst, uint8_t start_address, uint8_t num){

    uint8_t output[50] = {};
    if(num == 1)
        output[0] = start_address | L3GD20_READ | L3GD20_NO_AUTOINC;
    else
        output[0] = start_address | L3GD20_READ | L3GD20_AUTOINC;

    uint8_t input[50] = {};
    static_assert(sizeof(input) == sizeof(output));
    periph::blk::trx_transfer tran(output, input, num + 1);

    int ret = spi.transaction(0, tran);
    if(ret < 0)
        return -1;

    for(int i = 0; i < num; ++i)
        dst[i] = input[i + 1];

    return num;
}

int L3GD20::writeRegisters(const uint8_t* src, uint8_t s_address, uint8_t num)
{

    uint8_t output[50] {};
    if(num == 1)
        output[0] = s_address | L3GD20_WRITE | L3GD20_NO_AUTOINC;
    else
        output[0] = s_address | L3GD20_WRITE | L3GD20_AUTOINC;
    for(int i = 0; i < num; ++i)
        output[i + 1] = src[i];
    periph::blk::tx_transfer tran(output, num + 1);

    int ret = spi.transaction(0, tran);
    if(ret < 0)
        return -1;

    return num;
}

```

Rysunek 1. Metody realizujące transfery danych między MCU, a układem L3GD20

```
DEBUG CONSOLE  PROBLEMS 1  OUTPUT  TERMINAL

=====
| Angular rate [x, y, z]: [ -0.6104 deg/s,  0.2441 deg/s, -0.9766 deg/s] |
=====

=====
| Angular rate [x, y, z]: [ -0.1221 deg/s,  0.4883 deg/s, -0.3052 deg/s] |
=====

=====
| Angular rate [x, y, z]: [ -0.3052 deg/s,  0.1221 deg/s, -0.3662 deg/s] |
=====

=====
| Angular rate [x, y, z]: [ -0.1221 deg/s,  0.2441 deg/s, -0.4883 deg/s] |
=====

=====
| Angular rate [x, y, z]: [  0.4883 deg/s,  0.4883 deg/s, -0.9156 deg/s] |
=====
```

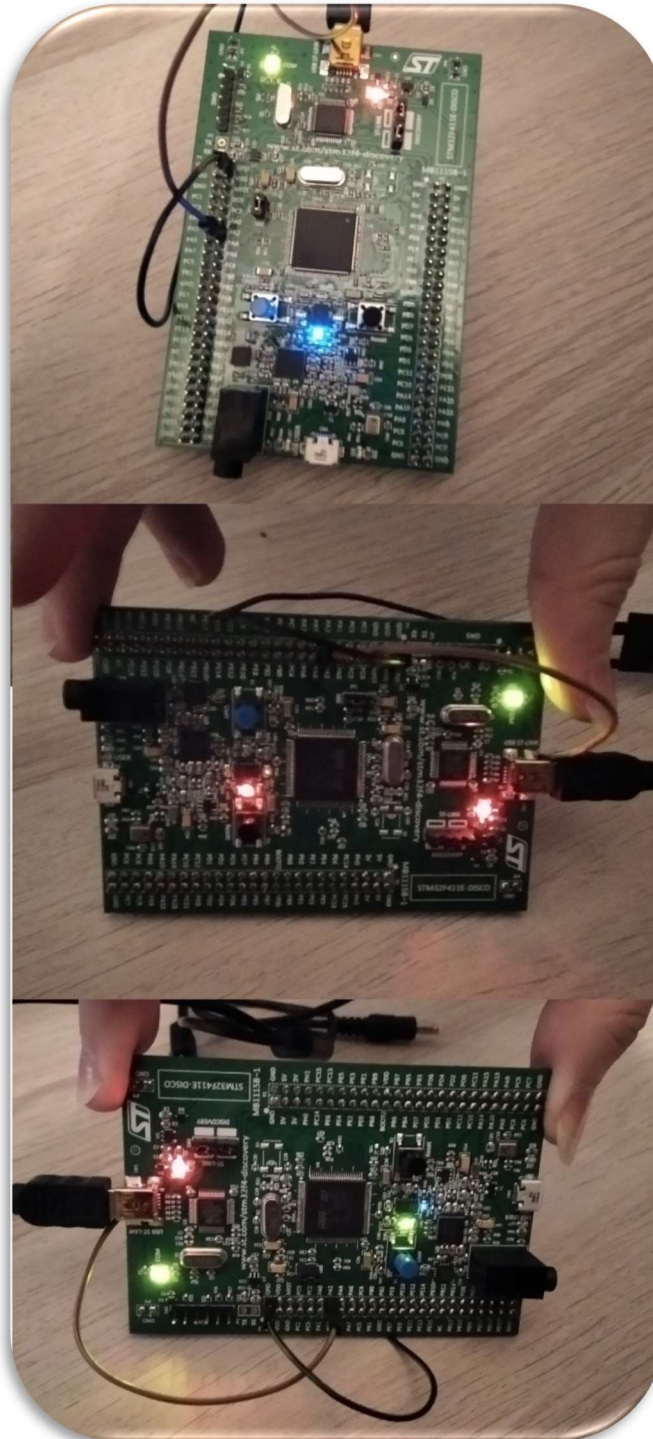
Rysunek 2. Wartości pomiarów wysyłane przez MCU na port szeregowy

ZADANIE DRUGIE

Drugie zadanie polegało na zaimplementowaniu kolejnego interfejsu komunikacyjnego, tym razem pomiędzy mikrokontrolerem, a czujnikiem LSM303. Czujnik ten zawiera w sobie zarówno akcelerometr jak i czujnik natężenia pola magnetycznego. Oba czujniki pozwalają na dokonywanie pomiarów wzdłuż wszystkich trzech wymiarów rozpinających przestrzeń. Jak już wspomniano, układ ten może się komunikować z jednostką centralną przy użyciu szyny I^2C .

Tym razem, napisany program powinien mierzyć wartość mierzonego przyspieszenia wzdłuż wszystkich dostępnych osi z częstotliwością 2Hz oraz wypisywać informację o pomiarze w oknie konsoli, jeżeli wykryta została jego zmiana. Ponownie, w celu rozwinięcia koncepcji projektu wprowadzono dwa rozszerzenia:

1. Odczytywany jest zarówno pomiar z akcelerometru, jak i z magnetometru. Pomiaru te są wysyłane na port szeregowy niezależnie po wykryciu zmiany wartości pomiaru.
2. Pomiar akcelerometru wzdłuż osi X i Y wizualizowany jest na diodach LD3...LD6 w sposób analogiczny do tego z zadania pierwszego. Diody sterowane są przez sygnał PWM generowany w układzie TIM4, którego wypełnienie zostało zmapowane na połowę zakresu pomiarowego. Diody wskazują kierunek działania przyspieszenia. Pozwala to na przykład na łatwe oszacowanie odchylenia powierzchni, na której leży płytkę od poziomu.



Rysunek 1. Wizualizacja pomiaru przyspieszenia wzdłuż osi X i Y z wykorzystaniem diod LED

W celu obsługi sensora stworzona została klasa **LSM303**, której interfejs jest **niemal identyczny interfejsem klasy L3GD20**. Jedyna różnica polega na tym, że ze względu na fakt, iż układ integruje w sobie dwa niezależne czujniki, metody zapisujące i odczytujące wartości wewnętrznych rejestrów układu posiadają dodatkowy parametr będący adresem czujnika na magistrali I^2C . Ponownie metody interfejsu stanowią wrapper dla wewnętrznych metod realizujących formalną stronę komunikacji. Metody te ukazano poniżej.

```
int LSM303::readRegisters(uint8_t* dst, uint8_t addr, uint8_t start_address, uint8_t num){
    uint8_t output {};
    if(num == 1)
        output = start_address | LSM303_WRITE | LSM303_NO_AUTOINC ;
    else
        output = start_address | LSM303_WRITE | LSM303_AUTOINC ;
    periph::blk::trx_transfer tran(&output, dst, 1, num);

    return i2c.transaction(addr, tran);
}

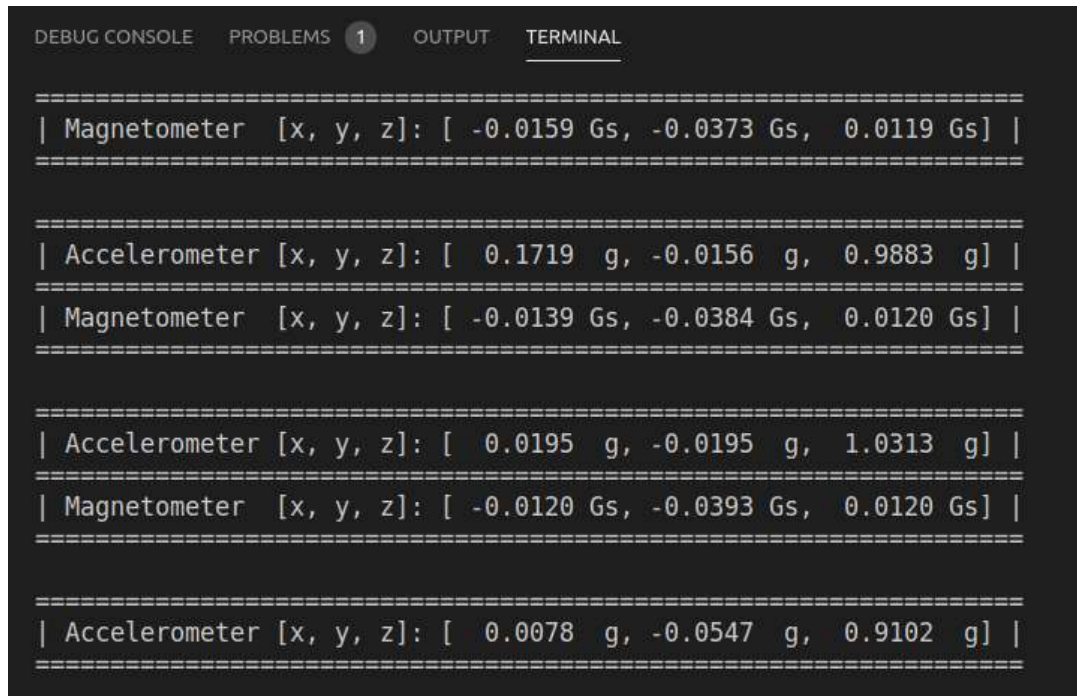
int LSM303::writeRegisters(const uint8_t* src, uint8_t addr, uint8_t start_address, uint8_t num){
    uint8_t output[7] {};
    if(num == 1)
        output[0] = start_address | LSM303_WRITE | LSM303_NO_AUTOINC ;
    else
        output[0] = start_address | LSM303_WRITE | LSM303_AUTOINC ;
    for(int i = 0; i < num; ++i)
        output[i + 1] = src[i];
    periph::blk::tx_transfer tran(output, num + 1);

    return i2c.transaction(addr, tran);
}
```

Rysunek 2. Prywatne metody klasy LSM303 realizujące transfery danych z wykorzystaniem magistrali I2C

Implementacja klasy zapewnia prostotę wymiany mechanizmów realizujących transfery danych. Sprowadza się to do zmiany implementacji metod `int readRegisters(uint8_t*`

`dst, uint8_t addr, uint8_t s_address, uint8_t num)` oraz `int writeRegisters(uint8_t* src, uint8_t addr, uint8_t s_address, uint8_t num)` wraz z konstruktorem klasy, który realizuje inicjalizację struktury komunikacyjnej.



The image shows a terminal window with a dark background and light-colored text. At the top, there are tabs for 'DEBUG CONSOLE', 'PROBLEMS' (with a '1' icon), 'OUTPUT', and 'TERMINAL' (which is selected). The terminal displays several lines of sensor data, each preceded by a separator line of equals signs. The data is organized into four groups, each containing an Accelerometer and a Magnetometer reading. The readings are formatted as '[x, y, z]: [value unit, value unit, value unit]'.

```
=====
| Magnetometer [x, y, z]: [ -0.0159 Gs, -0.0373 Gs,  0.0119 Gs] |
=====

=====
| Accelerometer [x, y, z]: [  0.1719 g, -0.0156 g,  0.9883 g] |
=====
| Magnetometer [x, y, z]: [ -0.0139 Gs, -0.0384 Gs,  0.0120 Gs] |
=====

=====
| Accelerometer [x, y, z]: [  0.0195 g, -0.0195 g,  1.0313 g] |
=====
| Magnetometer [x, y, z]: [ -0.0120 Gs, -0.0393 Gs,  0.0120 Gs] |
=====

=====
| Accelerometer [x, y, z]: [  0.0078 g, -0.0547 g,  0.9102 g] |
=====
```

Rysunek 3. Dane pomiarowe wyświetlane przez program poprzez port szeregowy