Obsługa magistrali I2C na STM32 od strony programowej przy pomocy bibliotek HAL jest bardzo podobna do obsługi magistrali SPI. Znowu obsługujemy 8 bitowe rejestry wewnątrz układu, które adresujemy przez 8 bitowe adresy **rejestrów**. Zasadnicza różnica jest w sposobie adresowania samego układu peryferyjnego. Na SPI wybieramy układ, z którym się będziemy komunikować przez ustawienie sygnału CS. Na I2C nadajemy adres **układu** po szynie danych, jeśli jakiś układ posiada taki adres to "odzywa się" poprzez nadanie (po tej samej szynie) sygnału ACK (od ang. *acknowledged*). Nie musimy aktywnie nasłuchiwać na sygnał ACK, robi to za nas kontroler magistrali I2C wewnątrz naszego mikrokontrolera STM32.

Przypomnienie obsługi transkacji na SPI

Zacznijmy zatem od przypomnienia obsługi magistrali SPI, żeby mieć grunt do porównania. Transakcje na magistrali SPI przeprowadzaliśmy w następujący sposób

1. Ustawialiśmy sygnał CS (Chip Select) podłączony do układu, z którym się komunikujemy w stan niski.
2. Nadawaliśmy za pomocą `HAL_SPI_Transmit()` adres rejestru układu peryferyjnego, który chcemy obsłużyć oraz ustawialiśmy bit rodzaju transakcji R/Wn (Read / Write(negated)). Dodatkowo mogliśmy ustawić bit autoinkrementacji adresów rejestrów w przypadku próby odczytu/zapisu bajtów z wielu kolejnych rejestrów naraz. Adres rejestru oraz bity R/Wn i autoinkrementacji ustawialiśmy w ramach jednego bajtu, podanego jako argument do `HAL_SPI_Transmit()`.
3. a) W przypadku transakcji odczytu, wywoływaliśmy `HAL_SPI_Receive()`. W przypadku odczytu wielu bajtów, ustawialiśmy bit autoinkremenctacji oraz zwiększaliśmy wartość odpowiadającego argumentu odpowiadającego za ilosć bajtów do odczytu w używanej funkcji.
   b) W przypadku transakcji zapisu, wywoływaliśmy `HAL_SPI_Transmit()`. W przypadku odczytu wielu bajtów, ustawialiśmy bit autoinkremenctacji oraz zwiększaliśmy wartość odpowiadającego argumentu odpowiadającego za ilosć bajtów do odczytu w używanej funkcji. Przy czym podpunkt (3.b) można pominąć i cały zapis zrealizować tylko w punkcie (2.) rozkazując nadanie wszystkich bajtów z bufor nadawczego.
4. Ustawialiśmy sygnał CS (Chip Select) podłączony do układu, z którym się komunikujemy w stan wysoki, co sygnalizowało zakończenie transkacji.

Celem powtórzenia, w skrócie:

1. Sygnał CS w stan niski.
2. `HAL_SPI_Transmit()` żeby określić rejestr i rodzaj transkacji.
3. a) `HAL_SPI_Receive()` żeby odebrać dane.
   b) `HAL_SPI_Transmit()` żeby zapisać dane.
4. Sygnał CS w stan wysoki.

Obsługa transkacji na I2C

Zasadnicza różnica między magistralą I2C a SPI będzie tutaj w sposobie wybierania układu z którym się chcemy skomunikować. I2C nie posiada sygnałów Chip Select. Nadajemy jedynie po linii danych adres układu, z którym się chcemy skomunikować (należy rozróżnić tutaj adres rejestru od adresu układu). **Adres I2C układu oraz jego rejestrów znajdują się w nocie katalogowej (datasheet) tegoż układu.**

W przypadku magistrali I2C transkacje przeprowadzać będziemy w następujący sposób:

1. a) `HAL_I2C_Mem_Write()` żeby zapisać coś do układu. Jako argumenty funkcji podajemy adres układu, adres rejestru, dane do zapisu (poprzez wskaźnik na bufor nadawczy), liczbę danych do zapisu. Bit R/Wn jest ustawiany automatycznie na zapis ze względu na wywoływaną funkcję `(...)_Write()`. W ramach bajtu będącego argumentem z adresem rejestru można też ustawić bit autoinkrementacji (należy odczytać w nocie katalogowej, który to będzie bit).
   b) `HAL_I2C_Mem_Read()` żeby odczytać coś z układu. Jako argumenty funkcji podajemy adres układu, adres rejestru, wskaźnik na bufor odbiorczy, liczbę danych do odczytu. Bit R/Wn jest ustawiany automatycznie na odczyt ze względu na wywoływaną funkcję `(...)_Read()`. W ramach bajtu będącego argumentem z adresem rejestru można też ustawić bit autoinkrementacji (należy odczytać w nocie katalogowej, który to będzie bit).

Tyle.

Przykłady

Przykład 1. Odczyt z rejestru

```
    #define I2C_RX_BUF_SIZE 2

    uint8_t i2c_rx_buf[I2C_RX_BUF_SIZE];

    uint8_t IC_addr = 0x06 << 1;  // Uwaga (a), wyjaśnione poniżej.
    uint8_t IC_example_reg = 0x26;

    uint8_t bytes_to_be_read = 1;

    HAL_I2C_Mem_Read(&hi2c1, IC_addr, IC_example_reg, 1, i2c_rx_buf,
bytes_to_be_read, 50);  // Uwaga (b)

    printf("IC_example_reg: 0x%02X\n", i2c_rx_buf[0]);
```

Uwagi:
a) funkcje `HAL_I2C_Mem_Read/Write()` oczekują - cytując - "left aligned address". Adresy I2C są standardowo 7-bitowe (co zresztą ustawiamy sami w graficznej konfiguracji urządzenia w pliku .ioc)
b) czwarty argument funkcji będziemy zawsze ustawiać na 1. Wyznacza on długość adresów w pamięci układu z którym się komunikujemy w bajtach. Wartość 1 oznacza 8 bitowe adresy.

Przykład 2. Odczyt wielu bajtów z kolejnych rejestrów

```
    #define I2C_RX_BUF_SIZE 6

    uint8_t i2c_rx_buf[I2C_RX_BUF_SIZE];

    uint8_t IC_addr = 0x06 << 1;
    uint8_t IC_example_reg = 0x26;

    uint8_t AUTO_INCREMEMT_ADDRESS = 0x01 << 7;  // Uwaga (c)
```

```
    uint8_t bytes_to_be_read = 6;

    HAL_I2C_Mem_Read(&hi2c1, IC_addr, IC_example_reg |
AUTO_INCREMENT_ADDRESS, 1, i2c_rx_buf, bytes_to_be_read, 50);

    printf("IC_example_reg1: 0x%02X, IC_example_reg2: 0x%02X, ...\n",
i2c_rx_buf[0], i2c_rx_buf[1], ...); // uwaga na pseudokod w tej linijce
```

Uwagi:
c) skąd wiemy, gdzie umieścić bit ustawiający auto-inkrementacje? Z datasheeta (noty katalogowej). Ta informacja zawsze znajduje się w sekcji opisującej komunikacje po I2C. Przykładowo w nocie katalogowej akcelerometru LSM303AGR (i jednocześnie magnetometru, bo znajdują się w tej samej obudowie), na stronie 38, czytamy:

> *After the start condition (ST) a slave address is sent, once a slave acknowledge (SAK) has been returned, an 8-bit subaddress (SUB) is transmitted: the 7 LSb represent the actual register address while the MSB enables address auto increment.* ***If the MSb of the SUB field is 1, the SUB (register address) is automatically increased to allow multiple data read/writes.***

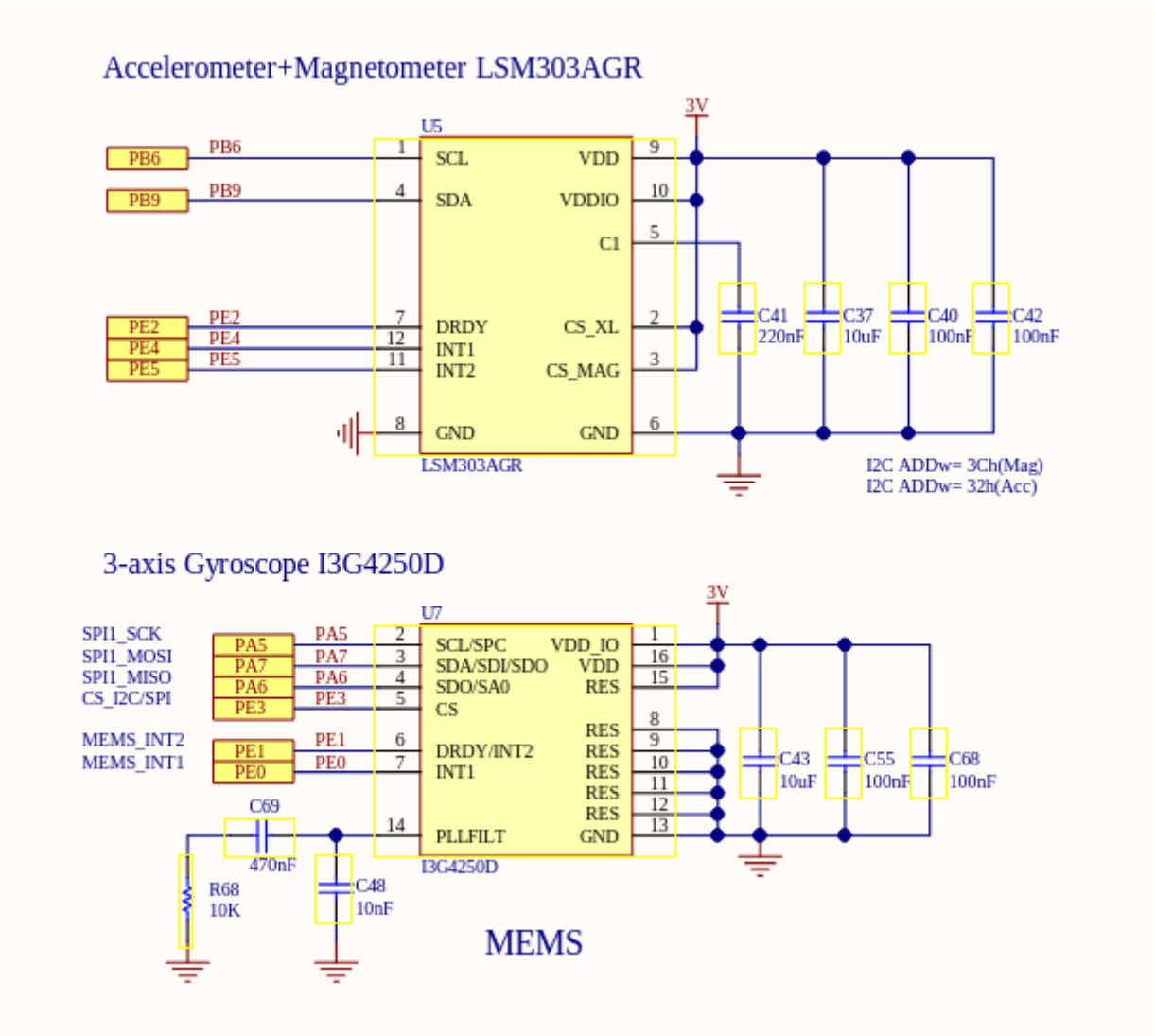
Przykład 3. Zapis do rejestru.

```
    #define I2C_TX_BUF_SIZE 2

    uint8_t i2c_tx_buf[I2C_TX_BUF_SIZE];

    uint8_t IC_addr = 0x06 << 1;
    uint8_t IC_example_reg = 0x26;

    i2c_tx_buf[0] = 0b11110111;

    HAL_I2C_Mem_Write(&hi2c1, IC_addr, IC_example_reg, 1, i2c_tx_buf, 1,
50);
```

Ustawienia w .ioc

Zanim naniesiemy ustawienia do kontrolera magistrali I2C, musimy się upewnić, że wyprowadzenia (tzw. piny) mikrokontrolera STM32 podłączone do magistrali I2C, na której znajduje się układ peryferyjny są odpowiednio skonfigurowane. Zaglądamy więc do schematu płytki DISCOVERY, z której korzystamy na

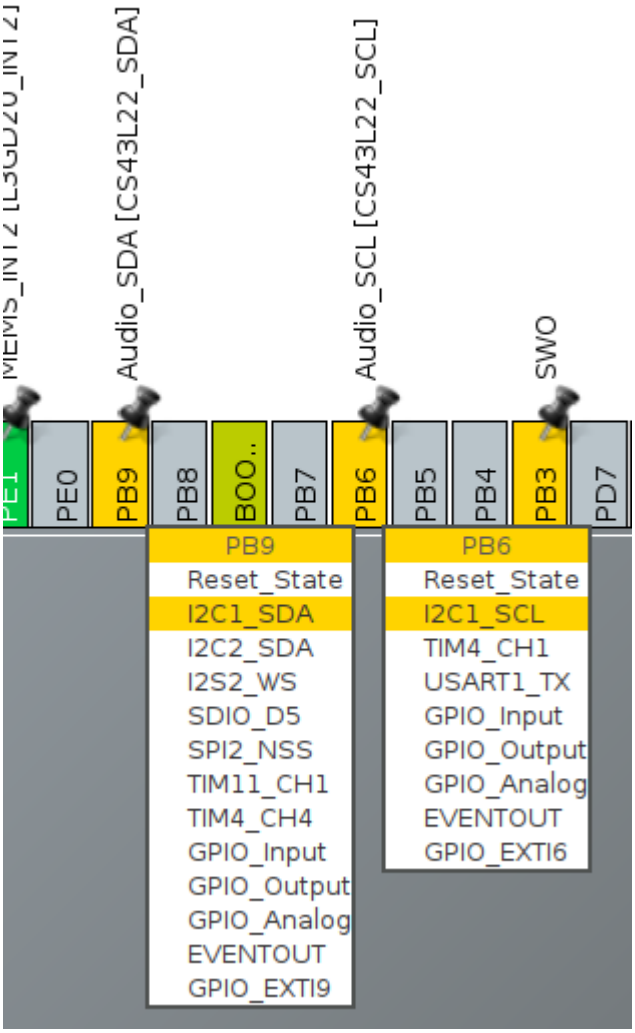
laboratoriach. Na jednej ze stron ze schematami widzimy:



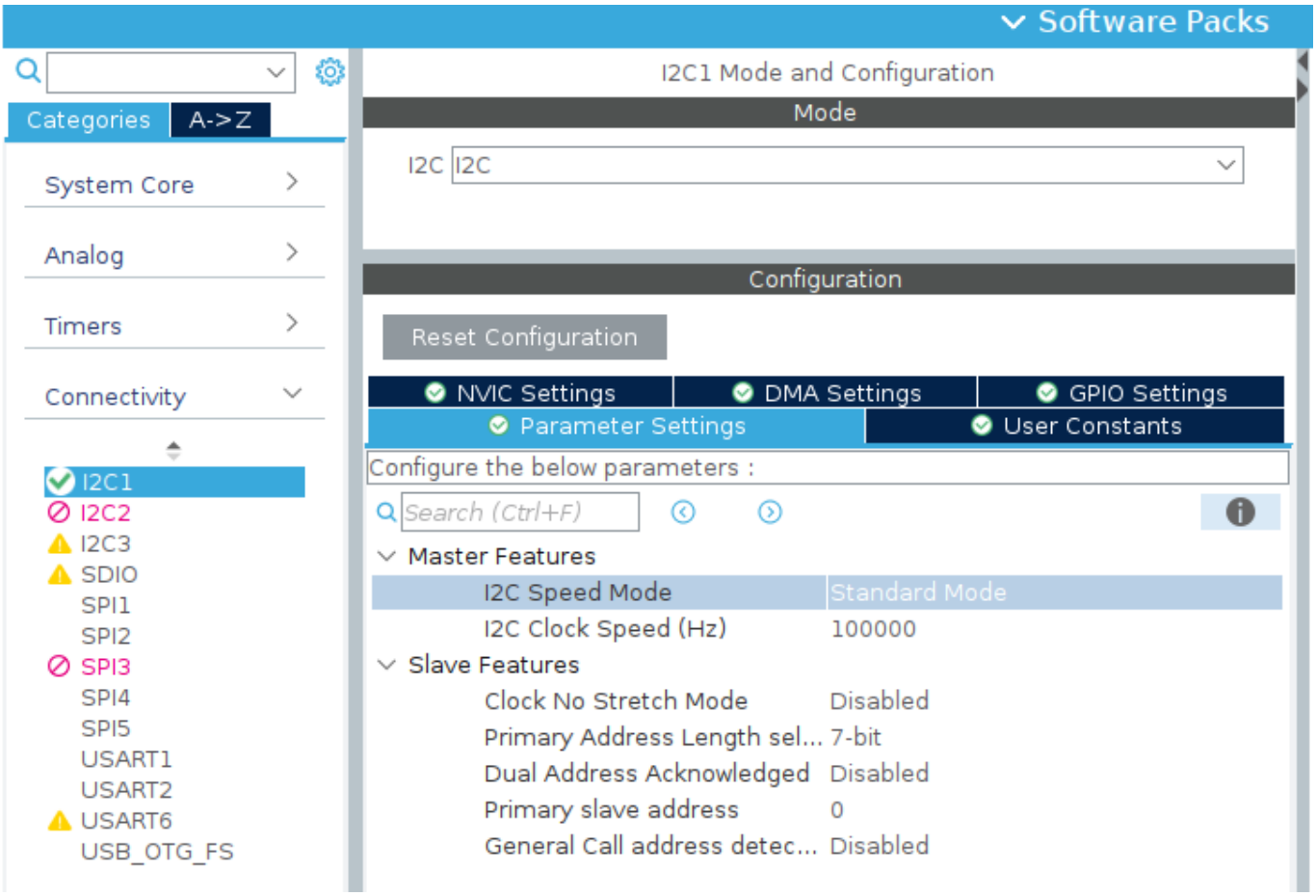
Zatem nasz akcelerometr ma piny magistrali I2C (SCL - Clock, zegar; SDA - Data, dane) podłączone do pinów PB6 i PB9 naszego mikrokontrolera STM32. Przy okazji widzimy też, że magistrala SPI żyroskopu poniżej podłączona jest do pinów PA5, PA7, PA6, PE3.

UWAGA: Niektóre płytki na laboratoriach korzystają z innego modelu żyroskopu: I3G4250D. Profesor Zabołotny ostrzegał przed tym w mailach, proszę je sobie ponownie przywołać. Najprościej zidentyfikować układ przez odczyt rejestru WHO_AM_I.

Wróćmy do nanoszenia ustawień I2C akcelerometru. Piny są już za nas wstępnie skonfigurowane.



Należy jedynie załączyć kontroler magistrali I2C.



W sekcji "Connectivity", klikamy na *I2C1*. W panelu "Mode" ustawiamy rubryczkę *I2C* z listy na *I2C*. Wszystkie ustawienia, które nam się następnie wyświetlą w panelu "Configuration" w zakładce "Parameter Settings" zostawiamy jako domyślne, tak jak na obrazku. Kontroler magistrali I2C jest gotowy do użycia.