Programowanie mikrokontrolerów 2.0

Magistrala I^2C, układy MEMS

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

9 listopada 2021

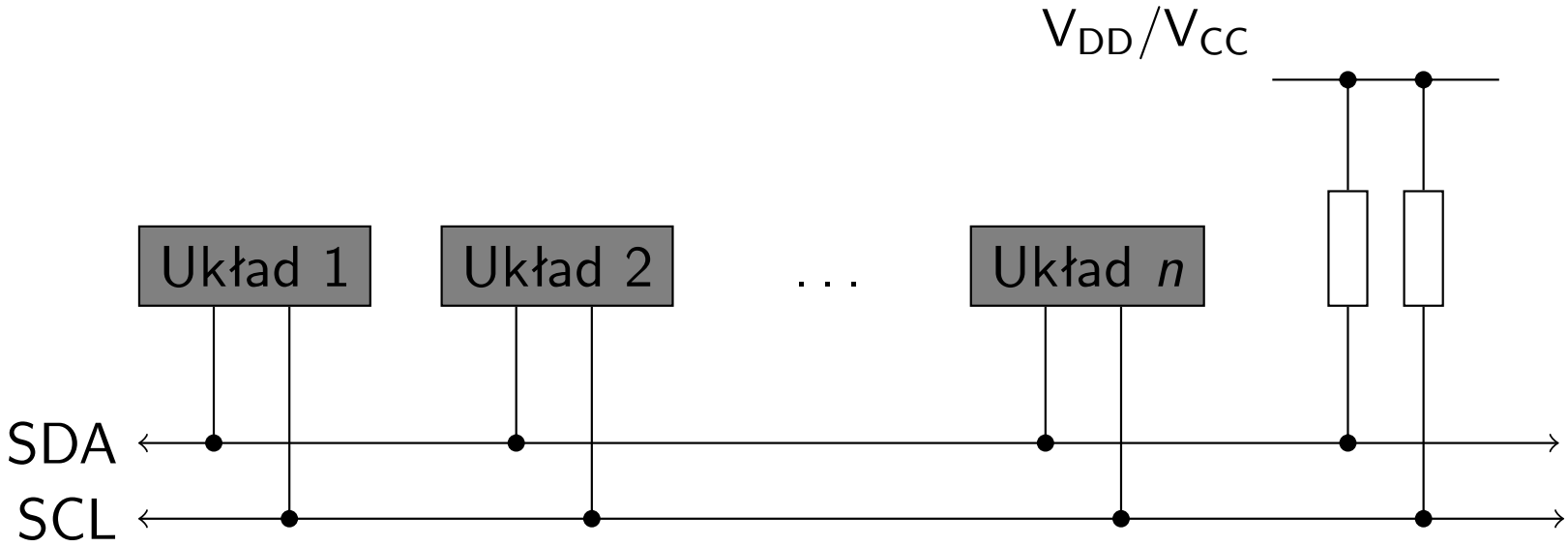
Magistrala I^2C

- Akronim Inter-Intergrated Circuit
- Opracowana w latach osiemdziesiątych XX wieku przez Philipsa
- Składa się z dwóch dwukierunkowych linii:
  - danych, SDA
  - zegara, SCL
- Transmisja danych odbywa się szeregowo i synchronicznie
- Do szyny może być przyłączonych wiele układów
- Identyfikacja układu odbywa się za pomocą jego adresu sprzętowego

Warianty magistrali I^2C

- Powstało kilka wersji I^2C:
  - wersja podstawowa (ang. *standard mode*) z taktowaniem do 100 kHz i 7-bitowymi adresami sprzętowymi (1982)
  - wersja 1.0 (ang. *fast mode*) z taktowaniem do 400 kHz i 10-bitową przestrzenią adresową (1992)
  - wersja 2.0 (ang. *high speed mode*) z taktowaniem do 3,4 MHz i rozszerzonym zakresem napięć (1998) oraz jej poprawiona wersja 2.1 (2000)
  - wersja 3.0 (ang. *fast mode plus*) z taktowaniem do 1 MHz (2007)
  - wersja 4.0 (ang. *ultra fast mode*) z taktowaniem do 5 MHz (2012)
  - wersja 5.0 (2012) oraz 6.0 (2014)
- System Management Bus
- Skupimy się na wersji podstawowej

Podłączenia elektryczne



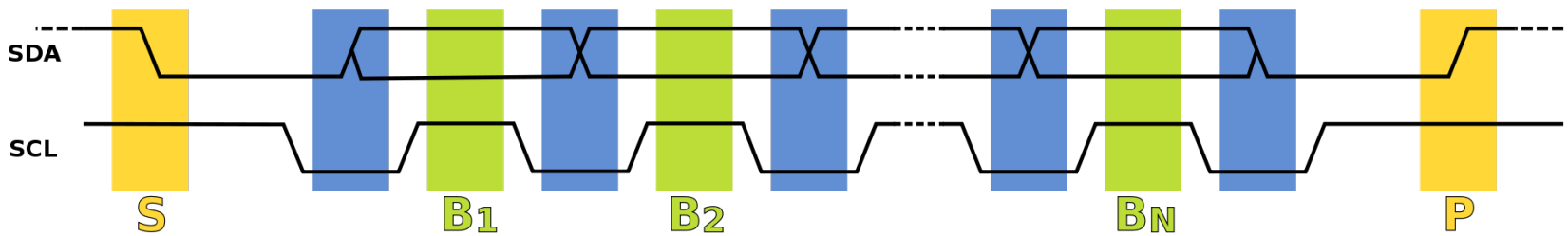
- Wyjścia układów są typu otwarty dren/kolektor
- Aby uzyskać na linii zero, należy podać na wyjście stan niski
- Aby uzyskać na linii jedynkę, wyjścia muszą być w stanie wysokiej impedancji, czyli muszą być odłączone od linii
- Aby linie nie „wisiały w powietrzu”, dołącza się zewnętrzne rezystory podciągające
- Stan linii jest iloczynem logicznym stanów wszystkich wyjść
- Wartości rezystorów zależą od liczby przyłączonych układów, szybkości transmisji i innych parametrów

Tryby pracy

- Każdy układ może pracować jako:
  - master (M) – inicjuje komunikację i generuje sygnał zegara
  - slave (S) – reaguje na dane pojawiające się na szynie
- Każdy układ może:
  - nadawać (T) – umieszczając dane na szynie
  - odbierać (R) – odczytując dane z szyny
- Mamy więc cztery tryby pracy: MT, MR, ST, SR
- W dalszym ciągu zakładamy, że tylko jeden układ pracuje w trybie master, choć protokół radzi sobie (przy pewnych dodatkowych założeniach) także z konfiguracjami z wieloma układami master

Protokół komunikacyjny

- Transmisja danych polega na nadaniu:
  - sygnału START
  - 9-bitowej ramki z adresem sprzętowym układu slave
  - pewnej liczby 9-bitowych ramek z danymi
  - sygnału STOP



Inicjowanie transmisji

- Transmisję inicjuje master, nadając sygnał START
- Od tej chwili szyna jest w stanie zajętości
- Master zwalnia szynę, nadając sygnał STOP
- Master może zainicjować kolejną transmisję bez zwalniania szyny, ponownie wysyłając sygnał START (tzw. REPEATED START)

Transmisja pojedynczej ramki

- Master generuje sygnał zegarowy na linii SCL
- W ośmiu kolejnych cyklach nadawca przesyła kolejne bity od najstarszego do najmłodszego
- W dziewiątym cyklu odbiorca generuje sygnał ACK lub NACK

Format ramki z adresem

- Składa się z 9 bitów:
  - 7-bitowy sprzętowy adres urządzenia slave
  - bit READ/WRITE:
    - 1 oznacza odczyt z urządzenia slave
    - 0 oznacza zapis do urządzenia slave
  - bit potwierdzenia – slave potwierdza odbiór własnego adresu, generując sygnał ACK w dziewiątym cyklu zegara
- Za generowanie sygnału zegara odpowiada master

Format ramki z danymi

- Składa się z 9 bitów:
  - 8 bitów danych wysyłanych przez nadawcę
  - bit potwierdzenia generowany przez odbiorcę, przyjmujący wartości:
    - ACK, jeśli odbiorca otrzymał dane i jest gotowy na następne
    - NACK, jeśli odbiorca nie może lub nie chce odbierać kolejnych danych
- Za generowanie sygnału zegara odpowiada master

Warstwa fizyczna

- Gdy szyna jest wolna, obie linie są w stanie wysokim
- Sygnał START polega na zmianie poziomu linii SDA z wysokiego na niski, przy jednoczesnym wysokim stanie linii SCL
- Poszczególne bity są ustawiane w fazie niskiej linii SCL i muszą pozostawać stabilne w fazie wysokiej
- Sygnał ACK polega na ustawieniu niskiego poziomu linii SDA w czasie dziewiątego cyklu zegara
- Sygnał NACK polega na pozostawieniu wysokiego poziomu linii SDA w czasie dziewiątego cyklu zegara
- Sygnał STOP polega na zmianie poziomu linii SDA z niskiego na wysoki, przy jednoczesnym wysokim stanie linii SCL

MEMS

- Akronim Microelectromechanical Systems

Źródło: http://www.memx.com

MEMS, przykładowe zastosowania

- Czujniki
  - akcelerometr
  - żyroskop
  - kompas
  - ciśnieniomierz
  - higrometr
  - mikrofon
  - ...
- Aktuatory
  - wyświetlacz DMD (ang. *digital micromirror device*)
  - przełącznik optyczny
  - mikrosilnik
  - mikrosiłownik
  - ...

Akcelerometr LIS35DE

- Współpracuje z magistralą I^2C lub SPI
- Dwie linie przerwań
- Dla programisty widoczny jako zestaw 8-bitowych rejestrów
  - główny rejestr konfiguracyjny `CTRL_REG1` ma numer `0x20`
  - rejestr konfigurujący przerwania `CTRL_REG3` ma numer `0x22`
  - rejestry `OUT_X`, `OUT_Y`, `OUT_Z` (o numerach `0x29`, `0x2B`, `0x2D`) zawierają aktualne wartości przyspieszenia odpowiednio dla osi X, Y, Z
- Dokumentacja `LIS35DE` oraz noty aplikacyjne `AN2335` i `AN2579` w katalogu `/opt/arm/stm32/doc`, aktualne wersje do ściągnięcia ze strony producenta `http://www.st.com`

Akcelerometr LIS35DE w zestawie laboratoryjnym

- Podłączony do magistrali I^2C
  - układ `I2C1`
  - linia SCL na wyprowadzeniu `PB8`, funkcja alternatywna
  - linia SDA na wyprowadzeniu `PB9`, funkcja alternatywna
  - adres sprzętowy `0x1C` lub `0x1D`, najmłodszy bit wybierany zworą SAD
  - rezystory podciągające linie SCL i SDA do zasilania są wlutowane na płytce
- Linie przerwań zewnętrznych
  - układ `EXTI`
  - linia INT1 na wyprowadzeniu `PA1`
  - linia INT2 na wyprowadzeniu `PA8`
  - na płytce nie ma rezystorów ustalających stan tych linii

Przykład konfigurowania magistrali I^2C

- Włącz taktowanie odpowiednich układów

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;
```

- Konfiguruj linię SCL na wyprowadzeniu `PB8`, a linię SDA – na wyprowadzeniu `PB9`, funkcja alternatywna

```
GPIOafConfigure(GPIOB, 8, GPIO_OType_OD,
                GPIO_Low_Speed, GPIO_PuPd_NOPULL,
                GPIO_AF_I2C1);
GPIOafConfigure(GPIOB, 9, GPIO_OType_OD,
                GPIO_Low_Speed, GPIO_PuPd_NOPULL,
                GPIO_AF_I2C1);
```

Przykład konfigurowania magistrali I^2C, cd.

- Konfiguruj szynę w wersji podstawowej

```
I2C1->CR1 = 0;
```

- Konfiguruj częstotliwość taktowania szyny

```
#define I2C_SPEED_HZ 100000
#define PCLK1_MHZ 16
I2C1->CCR = (PCLK1_MHZ * 1000000) /
            (I2C_SPEED_HZ << 1);
I2C1->CR2 = PCLK1_MHZ;
I2C1->TRISE = PCLK1_MHZ + 1;
```

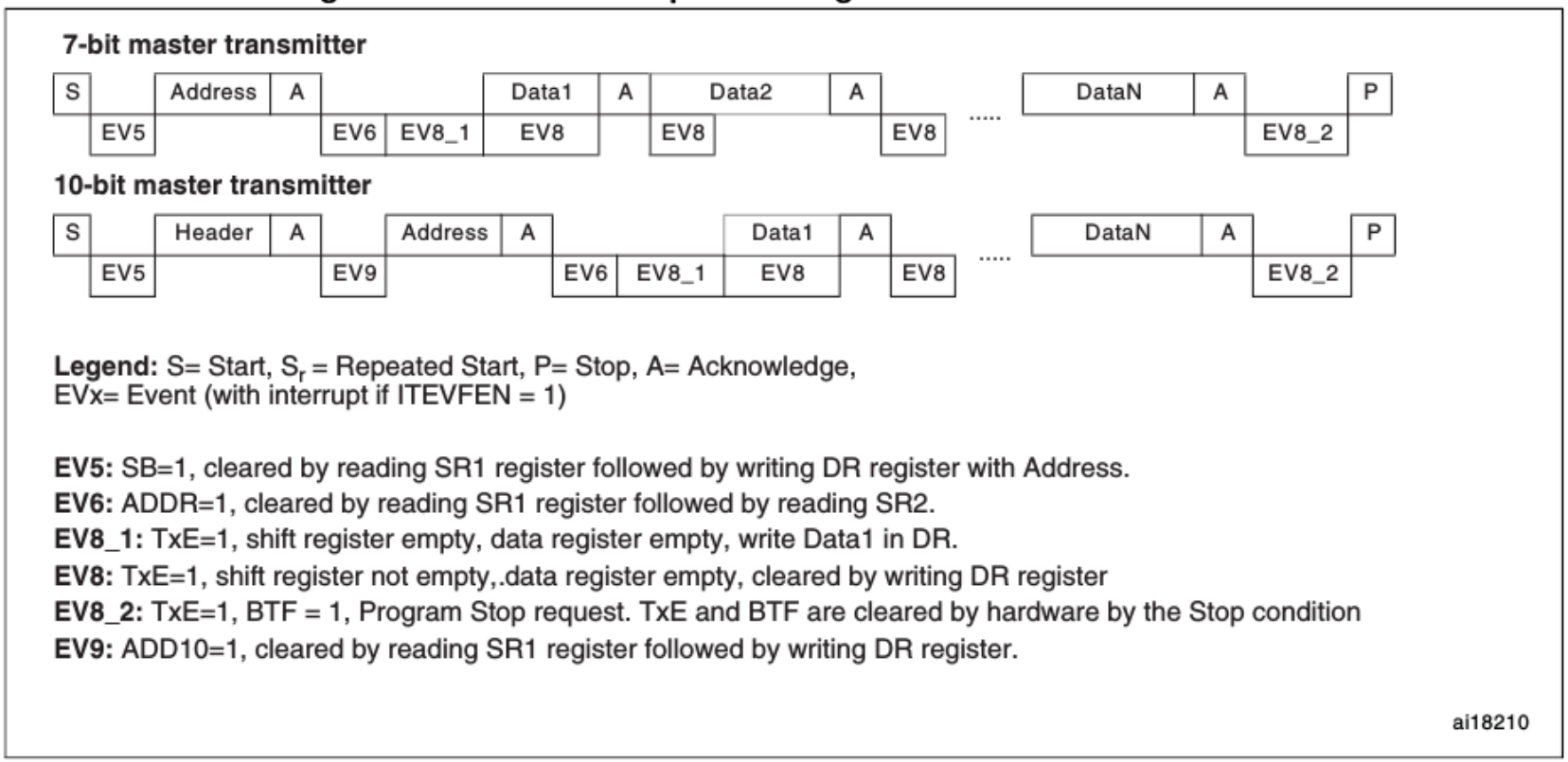
- Włącz interfejs

```
I2C1->CR1 |= I2C_CR1_PE;
```

- Rejestry układu `I2C1` są 16-bitowe

Tryb MT

Figure 164. Transfer sequence diagram for master transmitter



1. The EV5, EV6, EV9, EV8_1 and EV8_2 events stretch SCL low until the end of the corresponding software sequence.
2. The EV8 event stretches SCL low if the software sequence is not complete before the end of the next byte transmission.

Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Zapis do rejestru akcelerometru

- Zainicjuj transmisję sygnału START

  `I2C1->CR1 |= I2C_CR1_START;`

- Czekaj na zakończenie transmisji bitu START, co jest sygnalizowane ustawieniem bitu `SB` (ang. *start bit*) w rejestrze `SR1`, czyli czekaj na spełnienie warunku

  `I2C1->SR1 & I2C_SR1_SB`

- Zainicjuj wysyłanie 7-bitowego adresu slave'a, tryb MT

  `I2C1->DR = LIS35DE_ADDR << 1;`

- Czekaj na zakończenie transmisji adresu, ustawienie bitu `ADDR` (ang. *address sent*) w rejestrze `SR1`, warunek

  `I2C1->SR1 & I2C_SR1_ADDR`

- Skasuj bit `ADDR` przez odczytanie rejestru `SR2` po odczytaniu rejestru `SR1`

  `I2C1->SR2;`

Zapis do rejestru akcelerometru, dokończenie

- Zainicjuj wysyłanie 8-bitowego numeru rejestru slave'a

  ```
  I2C1->DR = reg;
  ```

- Czekaj na opróżnienie kolejki nadawczej, czyli na ustawienie bitu `TXE` (ang. *transmitter data register empty*) w rejestrze `SR1`, warunek

  ```
  I2C1->SR1 & I2C_SR1_TXE
  ```

- Wstaw do kolejki nadawczej 8-bitową wartość zapisywaną do rejestru slave'a

  ```
  I2C1->DR = value;
  ```

- Czekaj na zakończenie transmisji, czyli na ustawienie bitu `BTF` (ang. *byte transfer finished*) w rejestrze `SR1`, warunek

  ```
  I2C1->SR1 & I2C_SR1_BTF
  ```

- Zainicjuj transmisję sygnału STOP

  ```
  I2C1->CR1 |= I2C_CR1_STOP;
  ```

Wysyłanie ciągu bajtów po szynie I^2C, uwagi

- Jeśli chcemy wysłać wiecej bajtów, to po wstawieniu każdego z wyjątkiem ostatniego bajtu do rejestru `DR` czekamy na spełnienie warunku `I2C1->SR1 & I2C_SR1_TXE`, a po wstawieniu ostatniego czekamy na spełnienie warunku `I2C1->SR1 & I2C_SR1_BTF`
- Bit `TXE` jest ustawiany sprzętowo, gdy jednobajtowa kolejka nadawcza jest pusta
- Bit `TXE` jest kasowany programowo przez wstawienie bajtu do kolejki nadawczej, czyli zapisanie go do rejestru `DR`
- Bit `BTF` jest ustawiany sprzętowo, gdy ustawiony jest bit `TXE`, do rejestru `DR` nie został zapisany kolejny bajt do wysłania i zakończyła się transmisja aktualnie wysyłanego bajtu
- Bit `BTF` jest kasowany programowo przez ustawienie bitu `STOP` w rejestrze `CR1`

Wysyłanie ciągu bajtów po szynie I^2C, dalsze uwagi

- Nie należy czekać w nieskończoność na ustawienie bitu w rejestrze `SR1`
- Po przekroczeniu czasu oczekiwania należy zwolnić szynę, wysyłając sygnał STOP

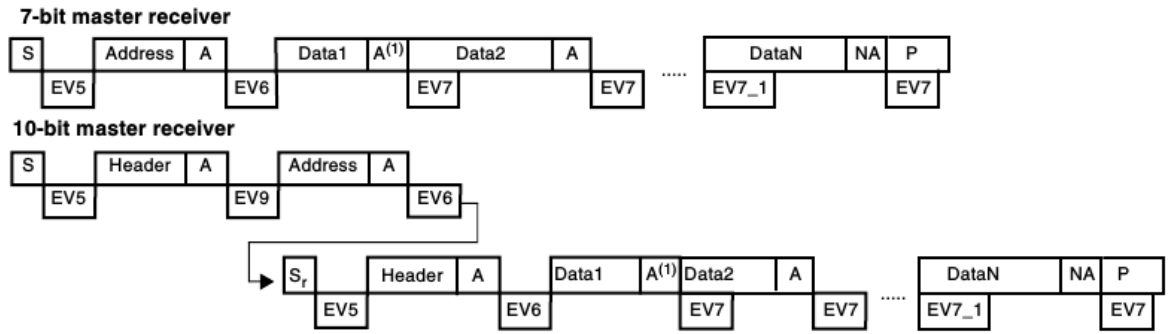
```
I2C1->CR1 |= I2C_CR1_STOP;
```

Zapis do rejestru akcelerometru, uwagi

- Nie wolno zapisywać do rejestrów, które w dokumentacji akcelerometru są oznaczone jako zarezerwowane

Tryb MR

Figure 165. Transfer sequence diagram for master receiver



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

EV5: SB=1, cleared by reading SR1 register followed by writing DR register.

EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with START = 1.
In case of the reception of 1 byte, the Acknowledge disable must be performed during EV6 event, i.e. before clearing ADDR flag.

EV7: RxNE=1 cleared by reading DR register.

EV7_1: RxNE=1 cleared by reading DR register, program ACK=0 and STOP request

EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

1. If a single byte is received, it is NA.
2. The EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. The EV7 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
4. The EV7_1 software sequence must be completed before the ACK pulse of the current byte transfer.

Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Odczyt z rejestru akcelerometru

- Zainicjuj transmisję sygnału START

  `I2C1->CR1 |= I2C_CR1_START;`
- Czekaj na ustawienie bitu `SB` w rejestrze `SR1`, warunek

  `I2C1->SR1 & I2C_SR1_SB`
- Zainicjuj wysyłanie 7-bitowego adresu slave'a, tryb MT

  `I2C1->DR = LIS35DE_ADDR << 1;`
- Czekaj na zakończenie transmisji adresu, warunek

  `I2C1->SR1 & I2C_SR1_ADDR`
- Skasuj bit `ADDR`

  `I2C1->SR2;`
- Zainicjuj wysyłanie numeru rejestru slave'a

  `I2C1->DR = reg;`
- Czekaj na zakończenie transmisji, czyli na ustawienie bitu `BTF` (ang. *byte transfer finished*) w rejestrze `SR1`, czyli na spełnienie warunku

  `I2C1->SR1 & I2C_SR1_BTF`

Odczyt z rejestru akcelerometru, cd.

- Zainicjuj transmisję sygnału REPEATED START

  ```
  I2C1->CR1 |= I2C_CR1_START;
  ```

- Czekaj na ustawienie bitu `SB` w rejestrze `SR1`, warunek

  ```
  I2C1->SR1 & I2C_SR1_SB
  ```

- Zainicjuj wysyłanie 7-bitowego adresu slave'a, tryb MR

  ```
  I2C1->DR = LIS35DE_ADDR << 1 | 1;
  ```

- Ustaw, czy po odebraniu pierwszego bajtu ma być wysłany sygnał ACK czy NACK
- Ponieważ ma być odebrany tylko jeden bajt, ustaw wysłanie sygnału NACK, zerując bit `ACK`

  ```
  I2C1->CR1 &= ~I2C_CR1_ACK;
  ```

- Czekaj na zakończenie transmisji adresu, warunek

  ```
  I2C1->SR1 & I2C_SR1_ADDR
  ```

- Skasuj bit `ADDR`

  ```
  I2C1->SR2;
  ```

Odczyt z rejestru akcelerometru, dokończenie

- Zainicjuj transmisję sygnału STOP, aby został wysłany po odebraniu ostatniego (w tym przypadku jedynego) bajtu

  `I2C1->CR1 |= I2C_CR1_STOP;`
- Czekaj na ustawienie bitu `RXNE` (ang. *receiver data register not empty*) w rejestrze `SR1`, warunek

  `I2C1->SR1 & I2C_SR1_RXNE`
- Odczytaj odebraną 8-bitową wartość

  `value = I2C1->DR;`

Odczyt ciągu bajtów po szynie I^2C, uwagi

- Jeśli chcemy odbierać więcej niż jeden bajt, to po zainicjowaniu wysyłania 7-bitowego adresu slave'a w trybie MR ustawiamy wysyłanie sygnału ACK, ustawiając bit `ACK`

  ```
  I2C1->CR1 |= I2C_CR1_ACK;
  ```

- Po odebraniu przedostatniego bajtu ustawiamy wysłanie sygnału NACK, zerując bit `ACK`

  ```
  I2C1->CR1 &= ~I2C_CR1_ACK;
  ```

- Inicjowanie transmisji sygnału STOP ustawiamy również po odebraniu przedostatniego bajtu

  ```
  I2C1->CR1 |= I2C_CR1_STOP;
  ```

- Nie należy czekać w nieskończoność na ustawienie bitu w rejestrze `SR1`
- Po przekroczeniu czasu oczekiwania należy zwolnić szynę, wysyłając sygnał STOP

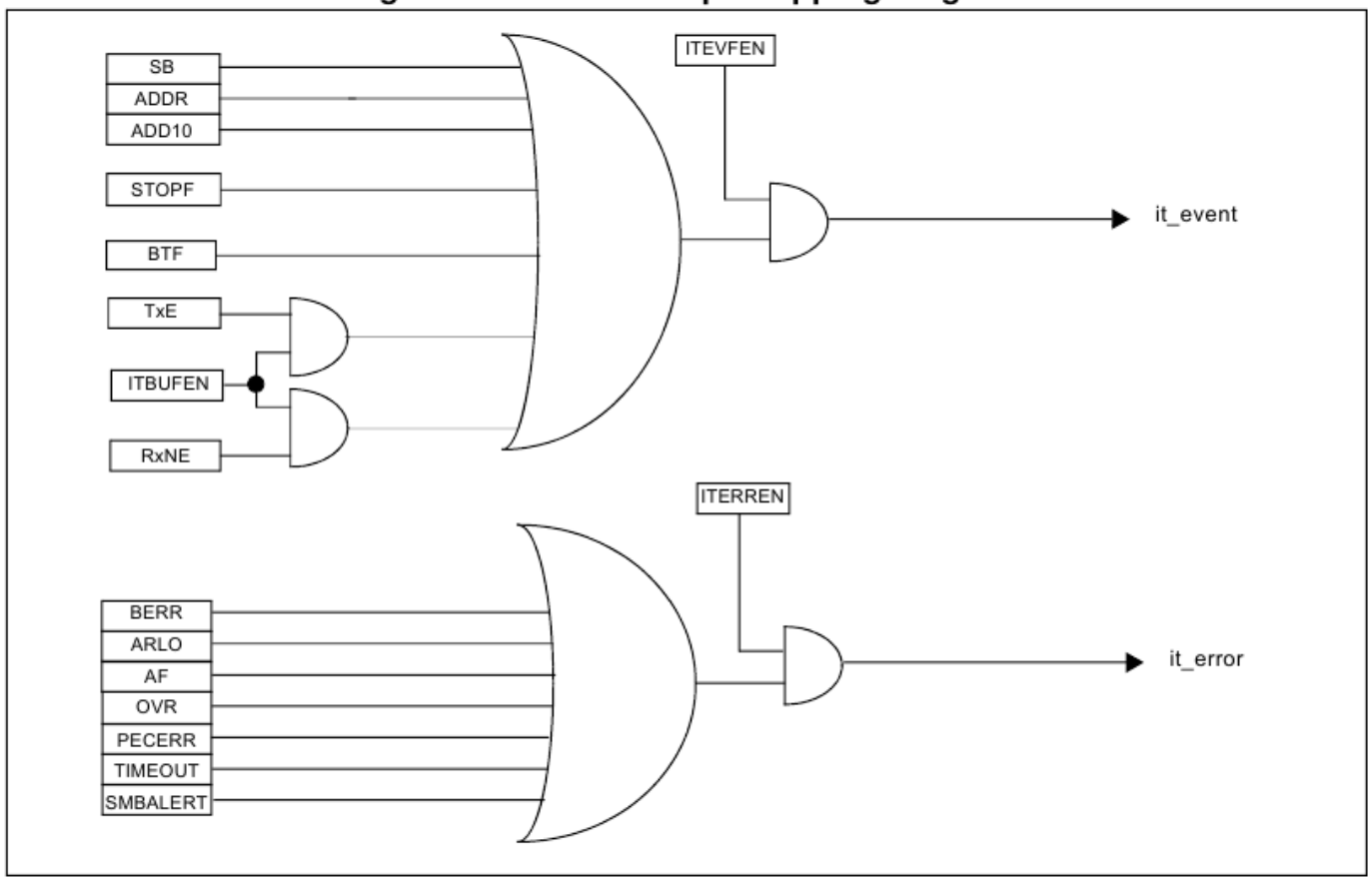
  ```
  I2C1->CR1 |= I2C_CR1_STOP;
  ```

Warstwa aplikacji I^2C

- Omówiony dotychczas protokół warstwy aplikacji I^2C nie jest jedynym spotykanym
- Żeby obsłużyć różne protokoły warstwy aplikacji I^2C, potrzebna jest procedura `i2c_write_read`, która najpierw zapisuje m bajtów, a potem odczytuje n bajtów, gdzie $m, n \geqslant 0$
- Zapisywane bajty zawierają polecenie, numer rejestru, wartość zapisywaną do rejestru itp.
- Odczytywane bajty zawierają wynik działania polecenia, wartość odczytaną z rejestru itp.
- Opisana wyżej procedura
  - zapisu do rejestru akcelerometru jest szczególnym przypadkiem procedury `i2c_write_read` dla $m = 2$ i $n = 0$
  - odczytu z rejestru akcelerometru jest szczególnym przypadkiem procedury `i2c_write_read` dla $m = n = 1$

Przerwania magistrali I^2C

Figure 166. I^2C interrupt mapping diagram



Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Przerwania I^2C

- Zamiast aktywnego oczekiwania na ustawienie bitów statusu można skorzystać z przerwań
- Mamy dwie procedury obsługi

```
void I2C1_EV_IRQHandler(void);
void I2C1_ER_IRQHandler(void);
```

- Przerwania aktywuje się za pomocą bitów `ITBUFEN`, `ITEVTEN`, `ITERREN` w rejestrze `CR2`
- Trzeba włączyć przerwania w układzie `NVIC`

```
NVIC_EnableIRQ(I2C1_EV_IRQn);
NVIC_EnableIRQ(I2C1_ER_IRQn);
```

- Przyczynę przerwania ustala się i kasuje za pomocą rejestrów `SR1` i `SR2`

Przerwania I^2C

- Przerwania `TxE` i `RxNE` są zgłaszane stanem odpowiednio kolejki nadawczej i odbiorczej, w odróżnieniu od np. przerwania `BTF` zgłaszanego zdarzeniem zakończenia transmisji
- Jeśli nie wypełnimy kolejki nadawczej (nie zapiszemy do rejestru `DR`), przerwanie `TxE` będzie ciągle zgłaszane
- Dlatego, jeśli nie mamy już nic do wysłania, należy to przerwanie wyłączyć (bit `ITBUFEN`)

Przerwania zewnętrzne

- Zamiast aktywnego sprawdzania wartości można zaprogramować akcelerometr, aby zgłaszał przerwanie
  - zakończenie pomiaru
  - przekroczenie wartości
  - pojedyncze lub podwójne kliknięcie
- Akcelerometr zgłasza przerwania poziomem, a STM32 obsługuje tylko przerwania zgłaszane zboczami
- Linia przerwania pozostaje w stanie aktywnym, dopóki zachodzi przyczyna przerwania
- Uwaga, akcelerometr może wykonać dwa kolejne pomiary, nie przestawiając między nimi linii przerwania w stan nieaktywny
- Przyczynę przerwania kasuje się przez wykonanie operacji na rejestrach akcelerometru, np. odczyt wyniku pomiaru, jeśli przyczyną przerwania jest zakończenie pomiaru

Przerwania zewnętrzne, cd.

- Konfigurowanie
  - wyłącz akcelerometr i przerwania (mogą pozostawać włączone po wyzerowaniu mikrokontrolera): wyzeruj rejestry `CTRL_REG1` i `CTRL_REG3` akcelerometru
  - skonfiguruj linię przerwania zewnętrznego i włącz przerwanie
  - skonfiguruj akcelerometr: ustaw stosowne wartości w rejestrach `CTRL_REG1` i `CTRL_REG3` akcelerometru
- Obsługa
  - skasuj bit w rejestrze `EXTI->PR`
  - zainicjuj komunikację z akcelerometrem kasującą przyczynę przerwania, np. zainicjuj odczyt danych
  - jeśli po zakończeniu linia przerwania nadal jest w stanie aktywnym, to oznacza, że ponownie wystąpiła przyczyna przerwania i trzeba tę sytuację obsłużyć

Zerowanie urządzeń podłączonych do szyny I^2C

- Zerowanie mikrokontrolera przyciskiem RESET nie zeruje akcelerometru
- Jeśli zaprogramowano mikrokontroler wadliwym programem, to zaprogramowanie poprawnym programem może nie uruchomić poprawnie akcelerometru – trzeba wyłączyć i włączyć zasilanie płytki

Ekspander z czujnikami

- Mamy kilka zestawów laboratoryjnych z czujnikami:
  - temperatury STLM75
  - wilgotności HTS221
  - ciśnienia LPS331AP
  - oświetlenia TSL2572
- Dokumentacja w katalogu `/opt/arm/stm32/doc`, aktualne wersje do ściągnięcia ze strony producenta `http://www.st.com`
- W zestawie podłączone do wspólnej szyny I^2C
- W mikrokontrolerze układ `I2C1` – ten sam, który obsługuje akcelerometr
- Kilka wskazówek na kolejnych slajdach

Czujnik temperatury STLM75

- Adres sprzętowy I^2C `0x48` lub `0x49` wybierany zworą JP1
- W najprostszym przypadku, aby odczytać wartość zmierzonej temperatury, czytamy dwa bajty
  - pierwszy bajt to 8-bitowa wartość ze znakiem zawierająca część całkowitą
  - najstarszy bit drugiego bajtu zawiera jeden bit części ułamkowej
- Możliwość pracy w trybie termostatu
  - ustawienie temperatury progowej i histerezy
  - wyjście sygnalizujące przekroczenie temperatury, podłączone do wyprowadzenia `PA8` mikrokontrolera

Czujnik wilgotności HTS221

- Adres sprzętowy I^2C `0x5F`
- Najprostsze użycie
  - konfigurowanie
    - ustaw konfigurację (rejestr `0x20`)
    - odczytaj wartości kalibracyjne (rejestry `0x30`, `0x31`, `0x36`, `0x37`, `0x3A`, `0x3B`)
    - oblicz współczynniki prostej interpolującej – patrz rys. 9 w dokumentacji
  - pomiar
    - odczytaj status (rejestr `0x27`)
    - jeśli wynik pomiaru jest dostępny, odczytaj go (rejestry `0x28`, `0x29`)
    - i oblicz względną wilgotność – patrz rys. 9 w dokumentacji
- Ma wyjście zgłaszające przerwanie, podłączone do wyprowadzenia `PA9` mikrokontrolera
- Mierzy też temperaturę

Czujnik ciśnienia LPS331AP

- Adres sprzętowy I^2C `0x5D`
- Najprostsze użycie
  - konfigurowanie
    - ustaw konfigurację (rejestr `0x20`)
  - pomiar
    - odczytaj status (rejestr `0x27`)
    - jeśli wynik pomiaru jest dostępny, odczytaj go (rejestry `0x28`, `0x29`, `0x2A`)
- Ma dwa wyjście zgłaszające przerwania, podłączone do wyprowadzeń `PA1` i `PA4` mikrokontrolera
- Mierzy też temperaturę

Czujnik oświetlenia TSL2572

- Adres sprzętowy I^2C `0x39`
- Najprostsze użycie
  - konfigurowanie
    - wyślij polecenie konfigurujące: 1. bajt to kod plecenia, 2. bajt to konfiguracja
    - oblicz współczynnik *lux per counts*
  - pomiar
    - odczytaj wyniku pomiaru: wyślij kod polecenia, odczytaj 4 bajty danych
    - oblicz wynik w luksach, posługując się wzorami podanymi w dokumentacji
- Ma wyjście zgłaszające przerwanie, podłączone do wyprowadzenia `PA0` mikrokontrolera