



| | | |
|---|--|---|
|  | Uniwersytet Technologiczno-Przyrodniczy im. J. i J. Śniadeckich w Bydgoszczy Wydział Telekomunikacji, Informatyki i Elektrotechniki |  |
| Przedmiot | Układy cyfrowe i mikroprocesory | |
| Prowadzący | Dr inż. Sławomir Bujnowski | |
| Temat projektu | Przygotowanie układu, który będzie sprawdzał temperaturę procesora oraz stężenie gazu ziemnego i LPG w powietrzu a następnie poprzez wyświetlacz LCD informował o zmierzonych wartościach. | |
| Student | Łukasz Farulewski | |

Celem projektu jest przygotowanie układu, który będzie sprawdzał temperaturę procesora oraz stężenie gazu ziemnego i LPG w powietrzu a następnie poprzez wyświetlacz LCD informował o zmierzonych wartościach.

Do tego należy zapewnić komunikację pomiędzy PC oraz MCU poprzez interfejs szeregowy USART. W tym celu zostanie zaprojektowany i zaimplementowany protokół komunikacyjny.

Protokół komunikacyjny

Komunikacja mikrokontrolera STM32F072RB z komputerem PC odbywać się będzie poprzez asynchroniczny szeregowy interfejs komunikacyjny **USART** z przerwaniami i z wykorzystaniem dwóch **buforów kołowych** (bez funkcji delay). Do transferu danych USART wykorzystuje 3 linie: sygnał **TxD** (transmisja danych), **RxD** (odbiór danych) oraz **GND** (masa).

W celu odczytania sygnałów analogowych pochodzących z czujników, zostanie obsłużony konwerter analogowo-cyfrowy **ADC** (Analog Digital Converter) dla 2 kanałów – czujnika gazu MQ5 oraz temperatury procesora (Temperature Sensor Channel). Mikrokontroler STM32F072 posiada wewnętrzny czujnik temperatury wbudowany w układ.

Na wejściu przetwornika ADC znajduje się multiplexer, dzięki któremu można mierzyć napięcie na kilkunastu kanałach – 16 zewnętrznych i 3 wewnętrznych z czego 2 zostaną wykorzystane na obsługę czujnika gazu oraz temperatury procesora.

ADC1:

- ✓ IN6 – czujnik gazu MQ5

✓ IN16 – Temperature Sensor Channel

W przypadku kanałów grupy podstawowej przetwornika ADC do dyspozycji jest tylko jeden, wspólny rejestr wyników (Data Register) dla wszystkich kanałów z grupy podstawowej (ADC1-> DR). Z tego powodu przy przetwarzaniu kilku kanałów z grupy podstawowej trudno określić skąd pochodzi odczytany w danym momencie wynik. W związku z tym zostanie użyty mechanizm **DMA** – Direct Memory Access z buforem cyklicznym i wszystkie otrzymane pomiary zostaną przesłane do tablicy w pamięci operacyjnej.

Konwersja ADC z DMA przebiega następująco:

1. aktywuję ADC oraz rozpoczynam konwersję używając funkcji HAL_ADC_StartDMA();
2. oczekuję na zakończenie konwersji w przerwaniu używając funkcji HAL_ADC_ConvCpltCallback() lub HAL_ADC_ConvHalfCpltCallback();
3. wyniki pomiarów są automatycznie przesyłane przez DMA do pamięci. DMA samodzielnie odczytuje pomiary i automatycznie wysyła je do dedykowanych rejestrów w pamięci operacyjnej.

Komunikacja MCU z wyświetlaczem LCD zostanie przeprowadzona poprzez interfejs **I2C**. Inter-Integrated Circuit – to magistrala łącząca układy scalone, służąca do transmisji szeregowej, synchronicznej, która odbywa się w trybie half-duplex. I2C używa dwóch przewodów do komunikacji – SDA oraz SCL.

Komunikacja MCU z czujnikiem gazu MQ5 prowadzona będzie na linii analogowej, gdzie wyjście analogowe czujnika połączone zostanie z przetwornikiem ADC skonfigurowanym na pinie PA6 (IN6) mikrokontrolera.

Transmisja i implementacja

Dane przesyłane są w formacie 8-bitowym z 1 bitem startu i stopu z prędkością transmisji 115200 b/s. 1 bit transmitowany jest w ciągu 1 / 115200 sekundy, czyli około 8 mikrosekund. Przy realizacji transmisji asynchronicznej z wykorzystaniem interfejsu szeregowego USART każdy bajt przesyłany jest niezależnie w osobnej ramce.

Budowa ramki:

| SOF | Payload | | EOF |
|------|--------------|----------------|------|
| 0x3B | Dane | Suma kontrolna | 0x3B |
| 1 B | 1-128 Bajtów | 1 B | 1 B |

Kodowanie ramki

Znakiem początku i końca ramki jest średnik – **0x3B (ASCII ;)**. Jeżeli w ładunku do wysłania pojawi się 1 bajt o wartości 0x3B (ASCII ;), to zostanie on zastąpiony przez 1 bajt **0x3F (ASCII ?)**. Jeśli jakkolwiek 1 bajt w ładunku do wysłania będzie o wartości 0x3F, to zostanie on zastąpiony przez **0x21,0x3F (ASCII !?)**.

Protokół komunikacyjny nie przewiduje wykorzystania innej konfiguracji znaków specjalnych w komendzie, dlatego próba zmiany przyjętego protokołu i próba jego implementacji w inny niż zaprojektowany sposób jest zabroniona.

W poniższym przykładzie komenda do wysłania z komputera PC do mikrokontrolera STM32 to **te?mp**. Zgodnie z przyjętym protokołem w pierwszej kolejności zostaje policzona suma kontrolna z komendy a następnie, przed wysłaniem ramki, komenda zostaje zakodowana i znak **0x3F (ASCII ?)** zostaje zakodowany na **0x21,0x3F (ASCII !?)**.

```
C:\Users\úukasz\Desktop\CRC_8\bin\Debug\CRC_8.exe
komenda: te?mp
payload: te!?mp | crc: 0xc9 | dlugosc: 7
frame: ;te!?mp; | crc: 0xc9 | dlugosc: 9

Process returned 0 (0x0)   execution time : 1.389 s
Press any key to continue.
```

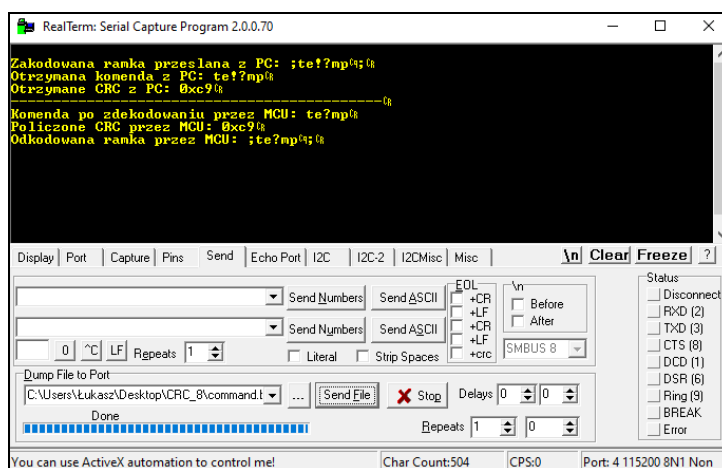
Konsola w środowisku Code::Blocks

Dekodowanie ramki

Jeżeli ramka została zakodowana po stronie PC, to po otrzymaniu ramki przez MCU, zostaje ona dekodowana zgodnie z przyjętym protokołem. To znaczy, jeżeli otrzymana ramka zawiera np. komendę **te!?mp**, to przechodzi ona przez proces dekodowania i do komputera PC zostaje zwrócona zdekodowana komenda **te?mp**.

Po otrzymaniu komendy **te!mp** mikrokontroler STM32 odsyła do komputera PC:

- zakodowaną ramkę przesłaną z PC
- otrzymaną komendę z PC
- otrzymane CRC z PC
- komendę po zdekodowaniu przez MCU
- policzone CRC przez MCU
- odkodowaną ramkę przez MCU

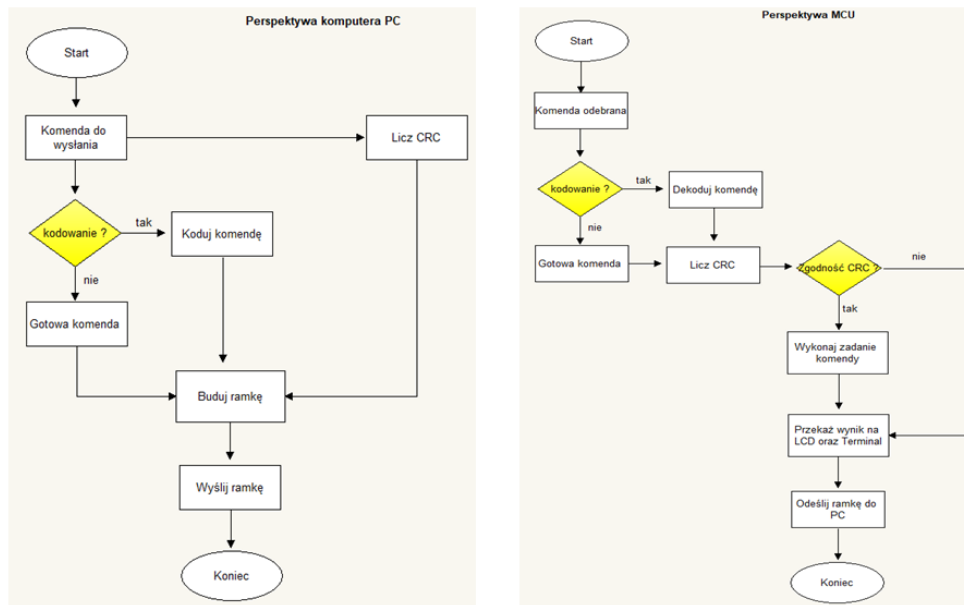


Kalkulacja CRC8

Wykorzystana suma kontrolna to CRC8 Dallas/Maxim i obliczana jest z pola danych dwukrotnie – przed wysłaniem ramki i po jej otrzymaniu. Suma kontrolna przed wysłaniem ramki obliczana jest przed procesem kodowania komendy. Oznacza to, że do zakodowanej komendy, jeżeli wymaga tego protokół, dołączana jest w ramce suma kontrolna niekodowanej komendy.

Po stronie mikrokontrolera zakodowana ramka musi zostać w pierwszej kolejności odkodowana. Następnie dla odkodowanej komendy liczona jest suma kontrolna. Tylko w takim przypadku obie sumy kontrolne mogą być porównywane.

Poniższe schematy w sposób orientacyjny lokalizują miejsce kalkulacji sumy kontrolnej dla wysyłanej i odbieranej komendy z perspektywy nadawcy – komputera PC oraz odbiorcy MCU.



Suma kontrolna obliczana jest w środowisku Code::Blocks i dołączana jest do pola danych. Payload ze znacznikami tworzy ramkę, która zapisywana jest w pliku command.bin na komputerze PC.

```

char CRC8(const char *data,int length)
{
    unsigned char crc = 0x00;
    unsigned char extract;
    unsigned char sum;
    int i;
    unsigned char tempI;

    for(i=0;i<length;i++)
    {
        extract = *data;
        for (tempI = 8; tempI; tempI--)
        {
            sum = (crc ^ extract) & 0x01;
            crc >>= 1;
            if (sum)
                crc ^= 0x8C;
            extract >>= 1;
        }
        data++;
    }
    return crc;
}

```

CRC-8 Dallas/Maxim

Kalkulacje CRC dla komendy **temp** w trybie debuggera w Code::Blocks. Dla wartości sum = 1, CRC jest XORowane z wielomianem 0x8C.

| | | | | | | | | | |
|-------------|--------------|-----|----|-----|----|-----|----|----|-----|
| crc >= 1 | crc | 0 | 0 | 0 | 70 | 35 | 87 | 43 | 21 |
| crc ^= 0x8C | crc (sum =1) | | | 140 | | 175 | | | 153 |
| | extract | 116 | 58 | 29 | 14 | 7 | 3 | 1 | 0 |
| | sum | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| | Templ | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | i | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Komendy funkcyjne

Poniższe komendy mają charakter funkcjonalny i mają za zadanie zwracać określone pomiary.

1. **htemp** oraz **temp** – pomiar temperatury w połowie (htemp) oraz na końcu bufora (temp). Komendy htemp oraz temp przesyłane są interfejsem szeregowym USART z PC do MCU tylko wówczas, gdy znajdują się między znacznikami – 0x3B, np. 0x3B tempCRC8 0x3B.
2. **hgaz** oraz **gaz** – pomiar stężenia gazu w połowie (hgaz) oraz na końcu bufora (gaz). Komendy hgaz oraz gaz przesyłane są interfejsem szeregowym USART z PC do MCU tylko wówczas, gdy znajdują się między znacznikami – 0x3B, np. 0x3B gazCRC8 0x3B.

Komendy dodatkowe

W projekcie wykorzystano także komendy ze znakami specjalnymi – ‘;’, ‘?’ oraz ‘!’, np. ‘te?mp’, ‘te;mp’. Komendy te nie zwracają wyników z żadnych pomiarów, są one jednak obsługiwane przez protokół komunikacyjny i w każdej chwili można przypisać im określoną funkcję – np. pomiar temperatury lub stężenia gazu. Po stronie komputera PC komendy te przechodzą przez proces kodowania i są wysyłane w ramce do mikrokontrolera. Przesłana ramka jest analizowana a następnie rozpakowywana. Komenda po stronie MCU zostaje odkodowana. Następnie mikrokontroler przekazuje komunikat na wyświetlacz LCD oraz na terminal komputera PC.

Analiza ramki

W terminalu RealTerm ładowany jest plik command.bin z ramką i ramka wysyłana jest z komputera PC do mikrokontrolera interfejsem szeregowym USART. Po odebraniu ramki przez mikrokontroler, ramka zostaje poddana analizie. Ramka jest sprawdzana pod kątem startu i końca ramki oraz rozmiaru payloadu. Do bufora zapisywane są dane oraz suma kontrolna.

```

uint8_t USART_getline(char * buff) {

    static uint8_t bf[129];
    static uint8_t idx = 0;
    static int sof_received = 0;
    int i;
    uint8_t ret;

    while (czyNiePustyRx()) {

        bf[idx] = USART_getchar();

        if (bf[idx] == 0x3B) {
            if (sof_received == 1) {
                bf[idx] = 0;
                for (i = 0; i <= idx; i++) {
                    buff[i] = bf[i];
                }
                ret = idx;
                idx = 0;

                sof_received = 0;

                return ret;
            }
            else {
                sof_received = 1;
            }
        }
        else if (sof_received == 1) {
            idx++;
            if (idx >= 129)
                idx = 0;
        }
    }
    return 0;
}

```

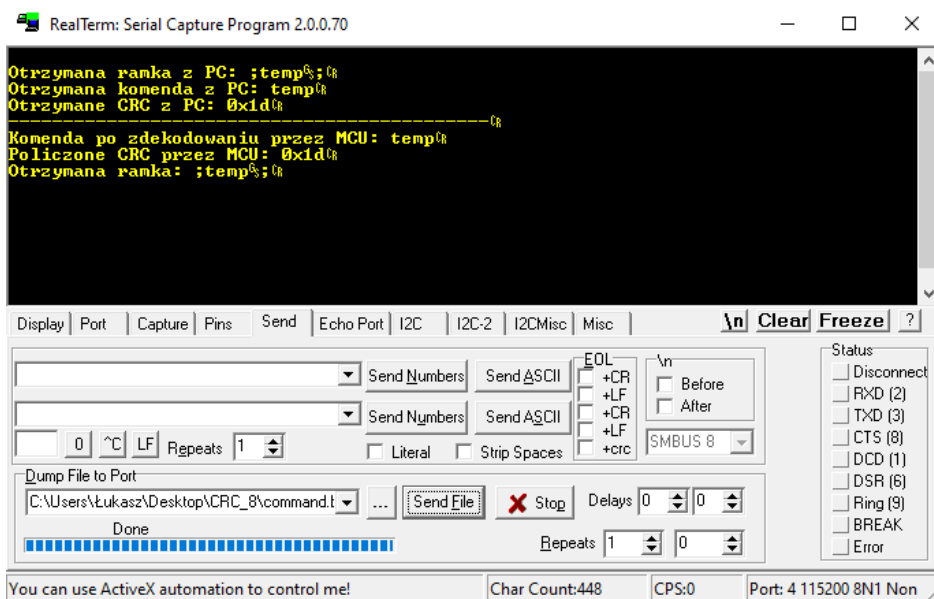
Jeżeli ramka spełnia wymagania protokołu, to zostaje rozpakowana – dane (komenda) oraz otrzymane CRC.

Jeżeli przesłana komenda została zakodowana przez PC, to zostaje zdekodowana przez mikrokontroler. Zdekodowana komenda jest następnie podstawą do policzenia nowej sumy kontrolnej po stronie STM32.

Jeżeli będzie zgodność CRC8 po obu stronach procesora, wówczas na wyświetlaczu LCD pojawi się pomiar temperatury lub stężenia gazu zgodnie z odebraną komendą.

Po odebraniu ramki, funkcją *fsend()* odsyłam otrzymaną ramkę z powrotem na terminal zgodnie z przyjętym protokołem: 0x3B DaneCRC8 0x3B.

Widok z terminalu po odesłaniu otrzymanej ramki.



W przypadku wartości ujemnych lub gdy wartość z przetwornika ADC od czujnika gazu odczyta pomiar stężenia tlenku węgla na poziomie np. 1500 ppm, wówczas zmienna zapisana będzie na 2 bajtach. W takim przypadku przed transmisją konieczne będzie podzielenie zmiennej na młodszy i starszy bajt.

| 1500 ppm => 2 bajty | | | | | | | | | | | | | | | |
|---------------------|-------|------|------|------|------|-----|-----|--------------|----|----|----|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| starszy bajt | | | | | | | | młodszy bajt | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Aby uzyskać wartość 1 bajtową w pierwszej kolejności wyciągam najstarszy bajt poprzez przesunięcie bitowe o 8 w prawo. Uzyskana wartość to: 00000101.

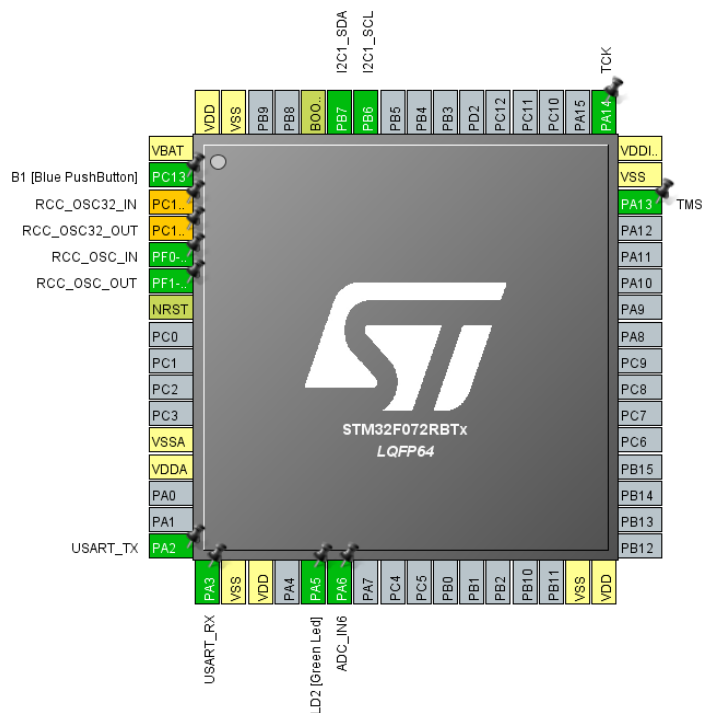
| | | | | | | | | | | | | | | | | |
|------|---------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| >> 8 | 1500 ppm => 2 bajty | | | | | | | | | | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

Następnie wyciągam najmłodszy bajt wykonując iloczyn logiczny. Uzyskany bajt to: 11011100.

| & | 1500 ppm => 2 bajty | | | | | | | | | | | | | | | |
|---|---------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

Konfiguracja mikrokontrolera w CubeMX.

Do komunikacji komputera PC z mikrokontrolerem STM32F072RB wykorzystano interfejs szeregowy USART na pinie PA2 (transmisja) oraz PA3 (odbiór). Wyświetlacz LCD 16x2 skomunikowano z mikrokontrolerem poprzez interfejs I2C, który skonfigurowano na pinach PB6 – linia zegara oraz PB7 – linia danych. W przypadku czujnika gazu MQ-5, wykorzystano pin PA6 mikrokontrolera, który skonfigurowano jako przetwornik ADC (IN6).



Podczas konfiguracji mikrokontrolera w CubeMX włączono domyślne peryferia oraz dodatkowo skonfigurowano następujące elementy:

Konfiguracja ADC

IN6: ✓

Temperature Sensor Channel: ✓

Sampling Time: 239,5

Continuous Conversion Mode: Enabled

Scan Conversion Mode: Forward

DMA Continuous Requests: Enabled

NVIC Settings: ✓

DMA w trybie bufora kołowego

Konfiguracja I2C

I2C: I2C

RCC: High Speed Clock: Cristal / Ceramic Resonator

LSE: Disabled

CRC: Activated

Przetwornik ADC skonfigurowano do pracy w trybie ciągłym (**continuous mode**) co powoduje, że po zakończeniu pomiaru w jednym kanale od razu rozpoczyna się kolejny pomiar w tym samym lub kolejnym kanale. Ponieważ przetwornik ma do obsługi 2 kanały analogowe, został włączony **scan mode** – czyli tryb automatycznego przechodzenia do pomiaru kolejnych kanałów. W praktyce funkcja ta jest już domyślnie włączona w mikrokontrolerze STM32F072, należy tylko wybrać tryb pracy – Forward – skanowanie kanałów od pierwszego do ostatniego lub Backward – skanowanie kanałów od ostatniego do pierwszego. Po przetworzeniu wszystkich kanałów, pomiar rozpoczyna się do nowa, od pierwszego kanału (Forward), czyli od IN6 – czujnika stężenia gazu.

Dla obu mierzonych kanałów analogowych ustawiono czas próbkowania (sampling time) na poziomie 239.5, co oznacza, że przy 12 bitowym przetworniku ADC (12,5 takta) pobranie jednego sygnału analogowego trwa 252 takty ADCCLK. Przy częstotliwości sygnału ADCCLK 14 MHz, czas przetwarzania oraz częstotliwość próbkowania przedstawia się następująco:

$T = (239,5 + 12,5) * 1 / 14\,000\,000 \text{ s} = 252 / 14\,000\,000 = 1 / 55\,555 \text{ s} = 18 \mu\text{s}$ (36 μs dla dwóch kanałów)

$f_s = 1 / T = 1 / 1/55\,555 = 1 * 55\,555 = 55\,555 = 55 \text{ kHz}$ (27 kHz dla dwóch kanałów)

Pomiar temperatury i stężenia gazu

Widok z debuggera Systemu Workbench for STM32 dla zadeklarowanej tablicy ADC_BUF[250][2].

| (x) Variables Breakpoints Expressions Registers Modules | | |
|---|-------------------|--------------------------|
| Expression | Type | Value |
| ADC_BUF | uint16_t [250][2] | 0x20000538 <ADC_BUF> |
| > [0...99] | uint16_t [100][2] | 0x20000538 <ADC_BUF> |
| > [100...199] | uint16_t [100][2] | 0x200006c8 <ADC_BUF+400> |
| > [200...249] | uint16_t [50][2] | 0x20000858 <ADC_BUF+800> |

| (*)= Variables Breakpoints Expressions Registers Modules | | |
|--|-------------------|-------------------------|
| Expression | Type | Value |
| ADC_BUF | uint16_t [250][2] | 0x20000538 <ADC_BUF> |
| [0...99] | uint16_t [100][2] | 0x20000538 <ADC_BUF> |
| ADC_BUF[0] | uint16_t [2] | 0x20000538 <ADC_BUF> |
| ADC_BUF[0][0] | uint16_t | 0 |
| ADC_BUF[0][1] | uint16_t | 1685 |
| ADC_BUF[1] | uint16_t [2] | 0x2000053c <ADC_BUF+4> |
| ADC_BUF[1][0] | uint16_t | 26 |
| ADC_BUF[1][1] | uint16_t | 1685 |
| ADC_BUF[2] | uint16_t [2] | 0x20000540 <ADC_BUF+8> |
| ADC_BUF[2][0] | uint16_t | 0 |
| ADC_BUF[2][1] | uint16_t | 1685 |
| ADC_BUF[3] | uint16_t [2] | 0x20000544 <ADC_BUF+12> |
| ADC_BUF[3][0] | uint16_t | 6 |
| ADC_BUF[3][1] | uint16_t | 1685 |
| ADC_BUF[4] | uint16_t [2] | 0x20000548 <ADC_BUF+16> |
| ADC_BUF[4][0] | uint16_t | 0 |
| ADC_BUF[4][1] | uint16_t | 1685 |
| ADC_BUF[5] | uint16_t [2] | 0x2000054c <ADC_BUF+20> |
| ADC_BUF[5][0] | uint16_t | 0 |
| ADC_BUF[5][1] | uint16_t | 1685 |
| ADC_BUF[6] | uint16_t [2] | 0x20000550 <ADC_BUF+24> |
| ADC_BUF[6][0] | uint16_t | 16 |
| ADC_BUF[6][1] | uint16_t | 1684 |
| ADC_BUF[7] | uint16_t [2] | 0x20000554 <ADC_BUF+28> |
| ADC_BUF[7][0] | uint16_t | 0 |
| ADC_BUF[7][1] | uint16_t | 1685 |
| ADC_BUF[8] | uint16_t [2] | 0x20000558 <ADC_BUF+32> |
| ADC_BUF[8][0] | uint16_t | 63 |
| ADC_BUF[8][1] | uint16_t | 1685 |

Odczytane wyniki to wartości po przetworzeniu przetwornika ADC1 bez kalibracji czujników. DMA skopiowało wartości pomiarów z rejestrów ADC do pamięci komputera. Odczyty dla stężenia gazu znajdują się w tablicy [250][2] na pierwszej pozycji indeksowanej [0], to znaczy [0][0], [1][0], [2][0], [3][0], [4][0], itd. Odczyty z czujnika temperatury znajdują się na drugiej pozycji – w tablicy znajdują się pod indeksem [1], to znaczy [0][1], [1][1], [2][1], [3][1], [4][1], itd. Łącznie 500 pomiarów, po 250 na każdy czujnik. Odczyty stężenia gazu oscylują wokół 0, ponieważ czujnik gazu podczas pomiaru nie był wystawiony na bezpośrednie działanie gazu.

Pomiar temperatury i stężenia gazu mierzony jest na przerwaniach w połowie oraz na końcu bufora. Ostateczne wyniki dla gazu to uśrednione wartości odczytane bezpośrednio z czujnika, natomiast w przypadku temperatury to przekazanie średniej wartości z pomiarów do wzoru na pomiar temperatury.

$$\text{Temperature (in } ^\circ\text{C)} = \frac{110\text{ }^\circ\text{C} - 30\text{ }^\circ\text{C}}{\text{TS_CAL2} - \text{TS_CAL1}} \times (\text{TS_DATA} - \text{TS_CAL1}) + 30\text{ }^\circ\text{C}$$

gdzie:

- TS_DATA to odczyt temperatury z czujnika – w tym przypadku jest to średnia wartość odczytana z pomiarów.
- TS_CAL 1 oraz TS_CAL2 to zmienne które przechowują adresy do kalibracji czujnika dla wartości 30 i 110 stopni C.

Wywołanie przerwania po wypełnieniu połowy bufora.

```
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc){  
    uint32_t sumHalfGas = 0;  
    uint32_t sumHalfTemp = 0;  
    float avgHalfTemp = 0.0;  
  
    for(int i=0; i<125; i++){  
        sumHalfGas += ADC_BUF[i][0];  
        sumHalfTemp += ADC_BUF[i][1];  
    }  
  
    avgHalfGas = sumHalfGas / 125;  
    avgHalfTemp = sumHalfTemp / 125;  
  
    HalfTemperature = (float)(110.0-30.0)/(float)(*TS_CAL2-*TS_CAL1)*(float)(avgHalfTemp - *TS_CAL1)+30.0;  
}
```

Odczyt temperatury, stężenia gazu oraz wartości pomocniczych z debuggera na przerwaniu w połowie bufora.

| | | |
|----------------------|----------|------------|
| (x)= sumHalfGas | uint32_t | 3693 |
| (x)= sumHalfTemp | uint32_t | 210698 |
| (x)= avgHalfTemp | float | 1685 |
| (x)= avgHalfGas | float | 29 |
| (x)= HalfTemperature | float | 44.4469528 |
| (x)= sumGas | uint32_t | 0 |
| (x)= sumTemp | uint32_t | 0 |
| (x)= avgGas | float | 0 |
| (x)= avgTemp | float | 0 |
| (x)= Temperature | float | 0 |

Wywołanie przerwania po wypełnieniu bufora.

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){  
    uint32_t sumGas = 0;  
    uint32_t sumTemp = 0;  
    float avgTemp = 0.0;  
  
    for(int i=125; i<250; i++){  
        sumGas += ADC_BUF[i][0];  
        sumTemp += ADC_BUF[i][1];  
    }  
  
    avgGas = sumGas / 125;  
    avgTemp = sumTemp / 125;  
  
    Temperature = (float)(110.0-30.0)/(float)(*TS_CAL2-*TS_CAL1)*(float)(avgTemp - *TS_CAL1)+30.0;  
}
```

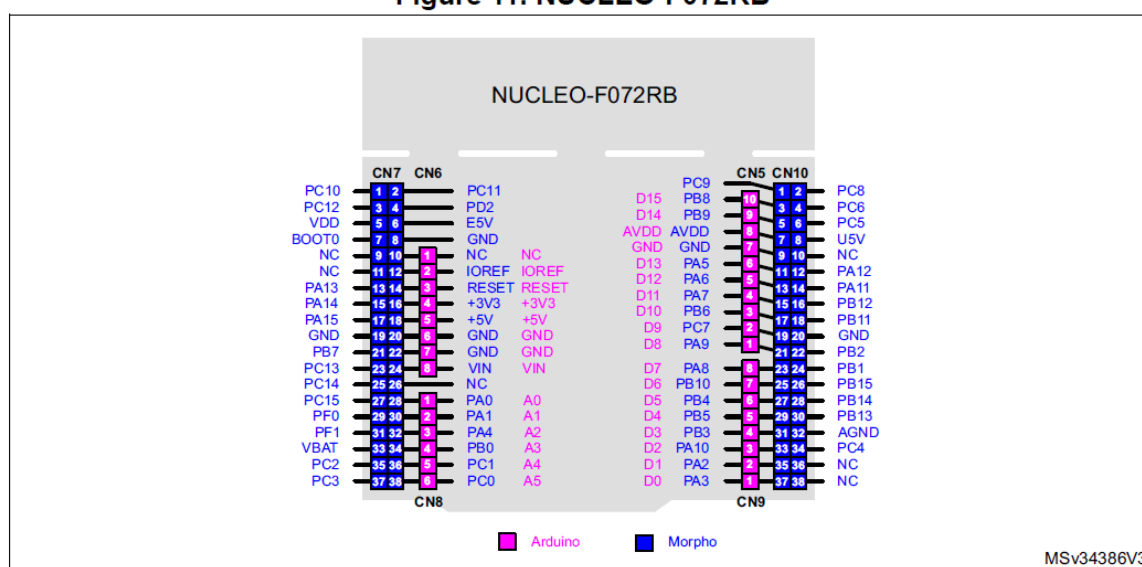
Odczyt temperatury, stężenia gazu oraz wartości pomocniczych z debuggera na przerwaniu po wypełnieniu bufora.

| | | |
|------------------|----------|------------|
| (*)= sumGas | uint32_t | 414 |
| (*)= sumTemp | uint32_t | 210620 |
| (*)= avgGas | float | 3 |
| (*)= avgTemp | float | 1684 |
| (*)= Temperature | float | 44,4469528 |

Informacje dodatkowe

1. Wyjścia mikrokontrolera STM32F072RB

Figure 11. NUCLEO-F072RB



2. Zastosowany mikrokontroler i peryferia w projekcie

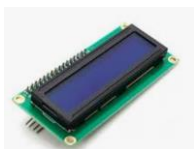
a) Mikrokontroler: STM32F072RB



b) Czujnik gazu MQ-5



c) Wyświetlacz LCD 2x16 znaków niebieski + konwerter I2C LCM1602



d) Przewody połączeniowe

