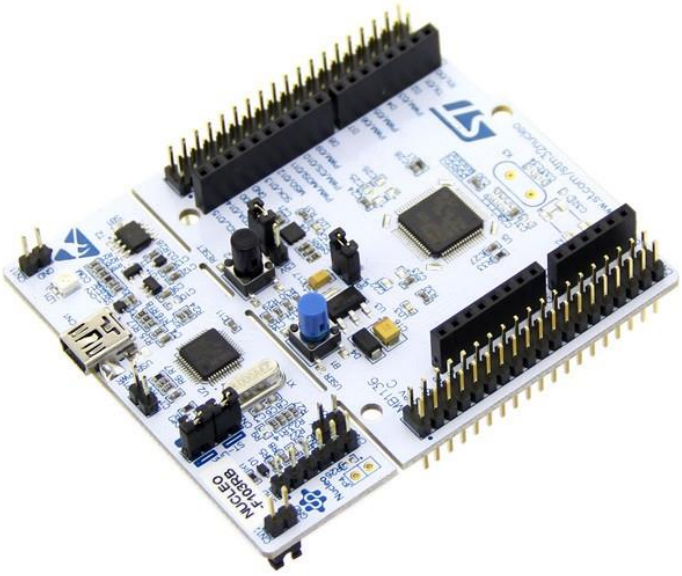
2019



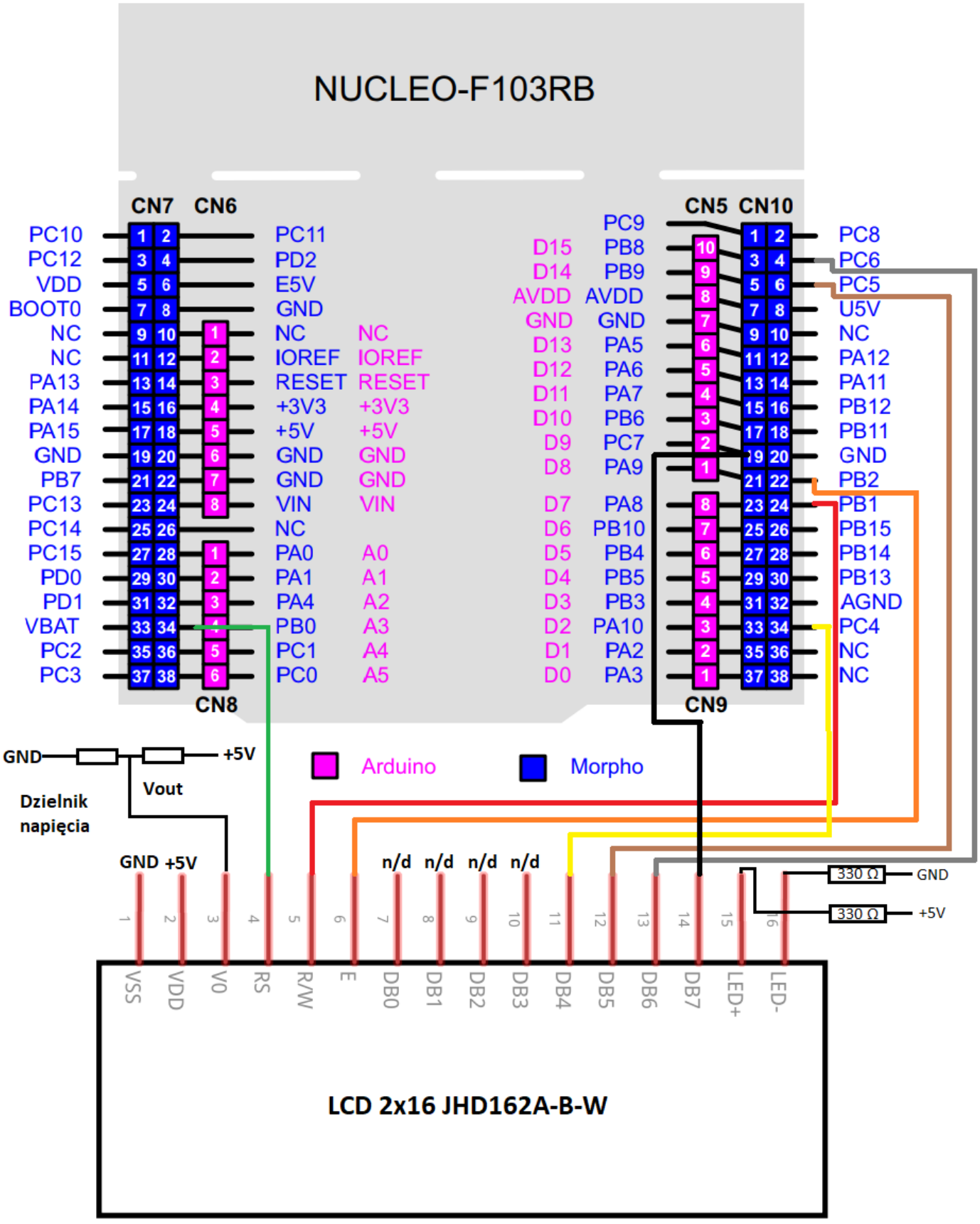
Czujnik cofania pojazdów

SULECKI_111010_UCIM_ST_18_19

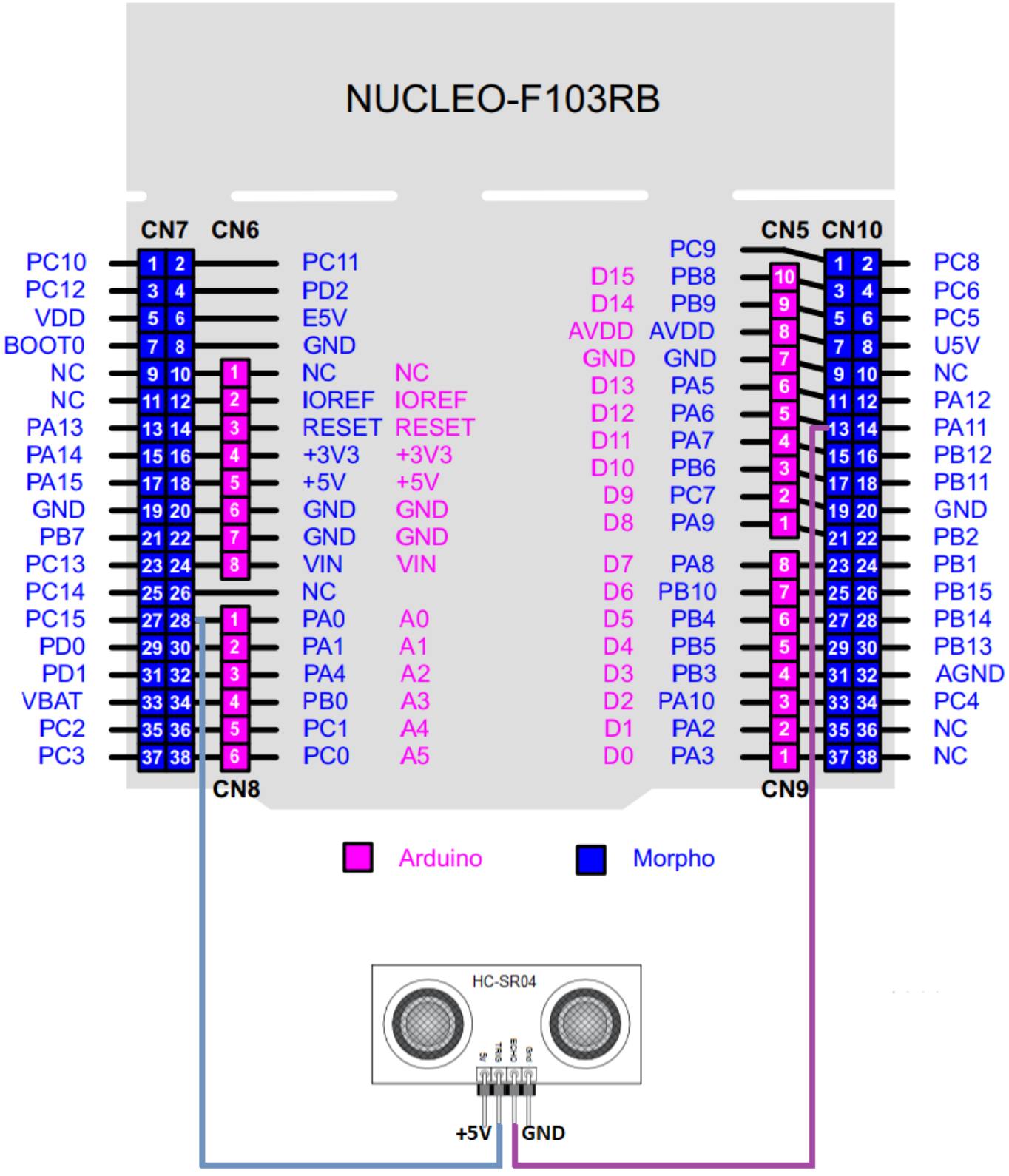
SPIS TREŚCI

SCHEMAT FIZYCZNEGO POŁĄCZENIA ELEMENTÓW ELEKTRONICZNYCH

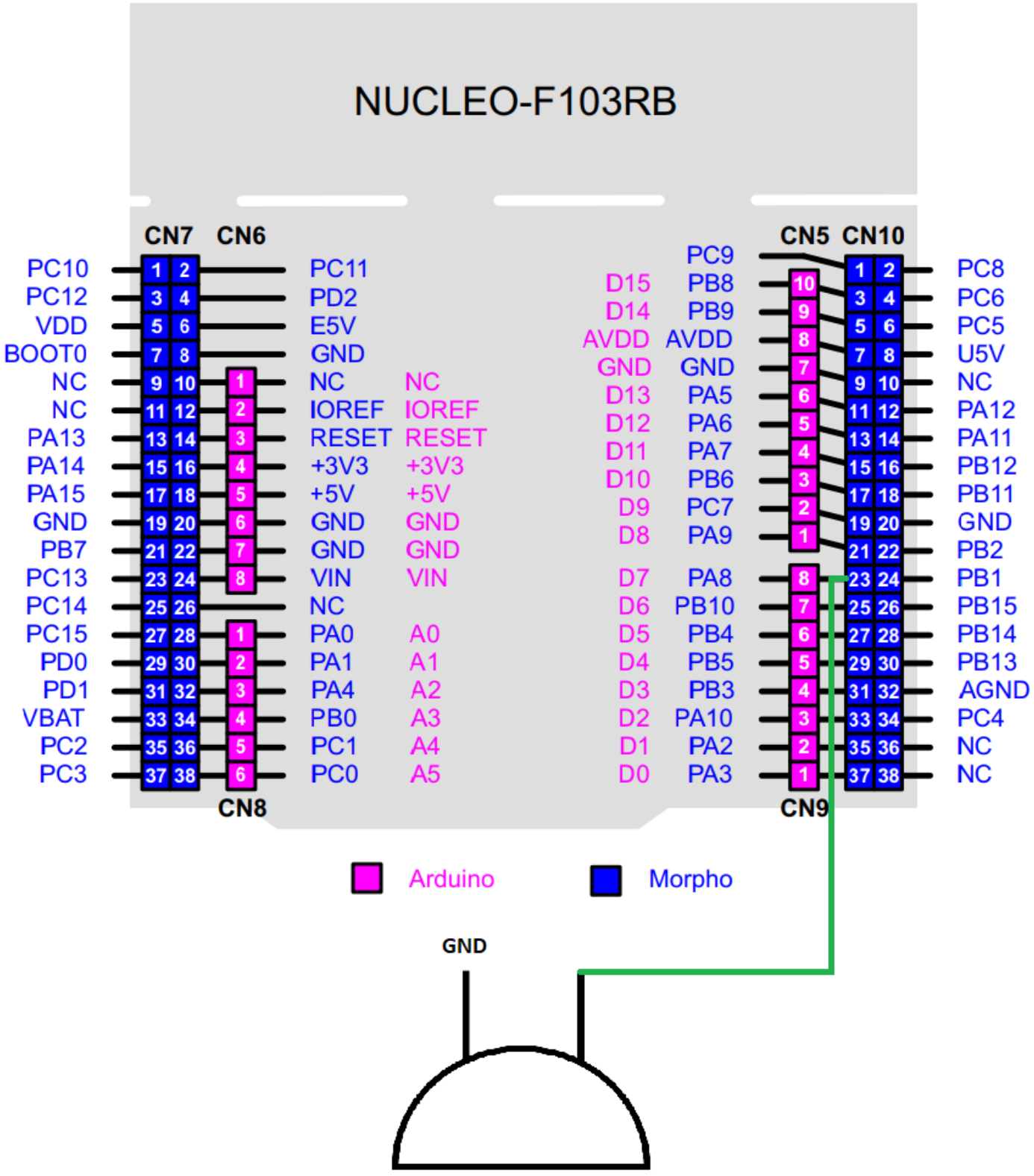
LCD



HC-SR04


NUCLEO-F103RB
CN7
CN6
CN5
CN10
CN8
CN9
PC10 PC12 VDD BOOT0 NC NC PA13 PA14 PA15 GND PB7 PC13 PC14 PC15 PD0 PD1 VBAT PC2 PC3
PC11 PD2 E5V GND NC IOREF RESET +3V3 +5V GND GND VIN NC PA0 PA1 PA4 PB0 PC1 PC0
NC IOREF RESET +3V3 +5V GND GND VIN A0 A1 A2 A3 A4 A5
PC9 D15 PB8 D14 PB9 AVDD AVDD GND GND D13 PA5 D12 PA6 D11 PA7 D10 PB6 D9 PC7 D8 PA9 D7 PA8 D6 PB10 D5 PB4 D4 PB5 D3 PB3 D2 PA10 D1 PA2 D0 PA3
PC8 PC6 PC5 U5V NC PA12 PA11 PB12 PB11 GND PB2 PB1 PB15 PB14 PB13 AGND PC4 NC NC
Arduino
Morpho
HC-SR04
+5V
GND

BRZĘCZYK PIEZOELEKTRYCZNY



KONFIGURACJA ORAZ SPOSÓB DZIAŁANIA UŻYTYCH ELEMENTÓW

LCD

LINIE STERUJĄCE

Kontrola LCD sprowadza się do ustawienia odpowiednich stanów logicznych na liniach wyświetlacza. Wyświetlacz posiada trzy linie sterujące:

- *RS (Register Select)* – określa czy w danym momencie operujemy na danych lub komendach wyświetlacza.
- *R/W (Read/Write)* – określa czy w danym momencie chcemy odczytać dane z wyświetlacza lub do niego zapisać.
- *E (Enable)* – Inicjowanie zapisu lub odczytu wyzwalane zboczem opadającym na tej linii.

Wyświetlacz posiada dwa rejestry, które są wybierane za pomocą linii sterującej *RS*:

- *IR (instruction register)* – przechowuje adresy dla pamięci *DDRAM* i *CGRAM*.
  - *DDRAM (display data RAM)* – przechowuje kody znaków, które chcemy wyświetlić.
  - *CGRAM (character generator RAM)* – przechowuje kody znaków zdefiniowanych przez użytkownika (max. 8).
- *DR (data register)* – przechowuje tymczasowo dane, które mają być odczytane lub wpisane do pamięci *DDRAM/CGRAM*.

Odpowiednia kombinacja linii *RS* i *R/W* oraz zainicjowana linią *E*, powoduje następujące działania:

RS	R/W	Operacja
0	0	Zapis do rejestru IR, wykonanie wewnętrznej komendy (np. czyszczenie ekranu)
0	1	Odczytanie bitu zajętości (linia DB7) i licznika adresów (linie DB0 do DB6)
1	0	Zapis do rejestru DR (z DR do DDRAM lub CGRAM)
1	1	Odczytanie z rejestru DR (z DDRAM lub CGRAM do DR)

Tabela 1. Kombinacja linii RS i R/W

MAKRA

W celu ułatwienia użytkowania i lepszej czytelności kodu, zdefiniowane są makra, które wykorzystywane są do kontrolowania LCD:

```
// GPIOx <--- ustawiamy A, B, C itp...
#define GPIOx GPIOC

// Linie danych
#define D4 GPIO_PIN_4
#define D5 GPIO_PIN_5
#define D6 GPIO_PIN_6
#define D7 GPIO_PIN_7

// LCD 2x16
#define LCD_ROWS 2
#define LCD_COLS 16

// Register Select
#define LCD_RS_PIN GPIO_PIN_0
#define LCD_RS_PORT GPIOB
```

```
// Read/Write
#define LCD_RW_PIN  GPIO_PIN_1
#define LCD_RW_PORT GPIOB

// Enable
#define LCD_E_PIN   GPIO_PIN_2
#define LCD_E_PORT  GPIOB

// Set and reset Register Select
#define LCD_RS_HIGH HAL_GPIO_WritePin(LCD_RS_PORT, LCD_RS_PIN, GPIO_PIN_SET);
#define LCD_RS_LOW HAL_GPIO_WritePin(LCD_RS_PORT, LCD_RS_PIN, GPIO_PIN_RESET);

// Set and reset Read/Write
#define LCD_RW_HIGH HAL_GPIO_WritePin(LCD_RW_PORT, LCD_RW_PIN, GPIO_PIN_SET);
#define LCD_RW_LOW HAL_GPIO_WritePin(LCD_RW_PORT, LCD_RW_PIN, GPIO_PIN_RESET);

// Set and reset Enable
#define LCD_E_HIGH HAL_GPIO_WritePin(LCD_E_PORT, LCD_E_PIN, GPIO_PIN_SET);
#define LCD_E_LOW HAL_GPIO_WritePin(LCD_E_PORT, LCD_E_PIN, GPIO_PIN_RESET);

//-------------------------------------------------------------------------

#define LCD_CLEAR                           0x01
#define LCD_HOME                            0x02
#define LCDC_ENTRY_MODE                     0x04
      #define LCD_EM_SHIFT_CURSOR               0x00
      #define LCD_EM_SHIFT_DISPLAY          0x01
      #define LCD_EM_LEFT                         0x00
      #define LCD_EM_RIGHT                        0x02
#define LCD_ONOFF                           0x08
      #define LCD_DISP_ON                       0x04
      #define LCD_CURSOR_ON                       0x02
      #define LCDC_CURSOR_OFF                     0x00
      #define LCDC_BLINK_ON                       0x01
      #define LCDC_BLINK_OFF                      0x00
#define LCD_SHIFT                           0x10
      #define LCDC_SHIFT_DISP                     0x08
      #define LCDC_SHIFT_CURSOR             0x00
      #define LCDC_SHIFT_RIGHT              0x04
      #define LCDC_SHIFT_LEFT                     0x00
#define LCD_FUNC                            0x20
      #define LCD_8_BIT                           0x10
      #define LCD_4_BIT                           0x00
      #define LCDC_TWO_LINE                       0x08
      #define LCDC_FONT_5x10                      0x04
      #define LCDC_FONT_5x7                       0x00
#define LCDC_SET_CGRAM                      0x40
#define LCDC_SET_DDRAM                      0x80

//Pierwsza linia wyswietlacza LCD
#define LCD_LINE1         0x00

//Druga linia wyswietlacza LCD
#define LCD_LINE2         0x40
```

ZAPISYWANIE

Wyświetlacz może działać w trybie 4 lub 8-bitowym, co oznacza, że w pierwszym trybie, aby przesłać pełną informację, należy uczynić to w dwóch turach po 4 bity. W projekcie został wykorzystany tryb 4-bitowy:

```
void LCD_writeHalf(uint8_t data) {
	LCD_E_HIGH;
	HAL_GPIO_WritePin(GPIOx, D4, (data & 0x01));
	HAL_GPIO_WritePin(GPIOx, D5, (data & 0x02));
	HAL_GPIO_WritePin(GPIOx, D6, (data & 0x04));
	HAL_GPIO_WritePin(GPIOx, D7, (data & 0x08));
	LCD_E_LOW;
}
```

Przeprowadzana jest operacja *AND* na znaku, który chcemy przesłać w celu ustawienia młodszej części bajtu. Na początku ustawiana jest linia *E* na stan wysoki a na końcu na stan niski, dzięki czemu inicjujemy operacje zboczem opadającym.

Następująca funkcja opakowuje ustawienie połowy bajta i pozwala na ustawienie całego bajta. Pierw ustawiane są 4 najbardziej znaczące bity a następnie 4 najmniej znaczące. Dodatkowo ustawiane są linie danych <D4, D7> w trybie wyjścia oraz pod koniec wysyłania bajtu sprawdzany jest bit zajętości, który omówiony jest w dalszej części dokumentacji:

```
void LCD_writeByte(uint8_t data) {
	LCD_pinsOutput();

	LCD_writeHalf(data >> 4);
	LCD_writeHalf(data);

	while(LCD_busyFlag() & 0x80);
}
```

Ostatecznie w celu wysłania danych wykorzystujemy funkcję, która ustawia linię *RS* w tryb danych oraz linię *R/W* w tryb zapisu:

```
void LCD_writeData(uint8_t data) {
	LCD_RS_HIGH;
	LCD_RW_LOW;
	LCD_writeByte(data);
}
```

Wysyłanie ciągu znaków, jest realizowane w następującej procedurze, przy wykorzystaniu zmiennej długości argumentów i formatowaniu danych:

```
void LCD_display(uint8_t x, uint8_t y, char *string, ...) {
	char BUFF_TMP[255];

	va_list valist;
	va_start(valist, string);
	vsprintf(BUFF_TMP, string, valist);
	va_end(valist);

	LCD_cursorPosition(x, y);
	for(int i = 0; BUFF_TMP[i] != '\0'; i++)
		LCD_writeData(BUFF_TMP[i]);
}
```

ODCZYTYWANIE

Odczytywanie danych również przeprowadzane jest w dwóch operacjach. W pierwszej kolejności odczyt 4 najstarszych bitów, następnie 4 najmłodszych:

```
uint8_t LCD_readHalf() {
        uint8_t tmp = 0;

        LCD_E_HIGH;
        tmp |= (HAL_GPIO_ReadPin(GPIOx, D4) << 0);
        tmp |= (HAL_GPIO_ReadPin(GPIOx, D5) << 1);
        tmp |= (HAL_GPIO_ReadPin(GPIOx, D6) << 2);
        tmp |= (HAL_GPIO_ReadPin(GPIOx, D7) << 3);
        LCD_E_LOW;

        return tmp;
}
```

Następna procedura odczytuje dane dla całego bajtu. W pierwszej kolejności odczytywany jest najbardziej znaczący półbajt, następnie odczytywany jest mniej znaczący:

```
uint8_t LCD_readByte() {
      uint8_t result = 0;

      LCD_pinsInput();
      LCD_RW_HIGH;

      result = (LCD_readHalf() << 4);
      result |= LCD_readHalf();

      return result;
}
```

BIT ZAJĘTOŚCI

Bit zajętości określa, czy LCD jest zajęty wykonywaniem jakiejś instrukcji lub nie. W tym celu stworzona została funkcja, która ustawia linię *RS* w tryb komend i odczytuje bit zajętości. Jest ona wykorzystywana w procedurze zapisywania bajtu danych a konkretnie w pętli *while*, która przestanie się wykonywać dopóki bit zajętości będzie równy zero:

```
uint8_t LCD_busyFlag() {
      LCD_RS_LOW;
      return LCD_readByte();
}

void LCD_writeByte(uint8_t data) {
      LCD_pinsOutput();

      LCD_writeHalf(data >> 4);
      LCD_writeHalf(data);

      while(LCD_busyFlag() & 0x80);
}
```

INICJALIZACJA

Inicjalizacja sprowadza się do wykonania pewnych wewnętrznych komend wyświetlacza. Są to:

- Ustawienie trybu 4-bitowego
- Ustawienie ilości wyświetlanych linii LCD
- Rozmiar czcionki
- Włączenie wyświetlacza
- Wyczyszczenie wyświetlacza
- Przesunięcie kursora na początek:

```
void LCD_init() {
	LCD_writeCmd(LCD_FUNC | LCD_4_BIT | LCDC_TWO_LINE | LCDC_FONT_5x7);
	LCD_writeCmd(LCD_ONOFF | LCD_DISP_ON);
	LCD_writeCmd(LCD_CLEAR);
	LCD_writeCmd(LCDC_ENTRY_MODE | LCD_EM_SHIFT_CURSOR | LCD_EM_LEFT);
}
```

WYŚWIETLANIE POMIARU

Ze względu na to, że LCD nie jest w stanie wyświetlać liczb zmiennoprzecinkowych należy przekonwertować wartość typu zmiennoprzecinkowego na ciąg znaków. Realizuje to procedura *ftoa* z następującymi parametrami:

- *Float n* – wartość wykonanego pomiaru
- *Char *res* – bufor na przekonwertowaną wartość
- *Int afterpoint* – ilość znaków po przecinku

```
void reverse(char *str, int len)
{
    int i=0, j=len-1, temp;
    while (i<j)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++; j--;
    }
}

// Converts a given integer x to string str[].  d is the number
 // of digits required in output. If d is more than the number
 // of digits in x, then 0s are added at the beginning.
int intToStr(int x, char str[], int d)
{
    int i = 0;
    while (x)
    {
        str[i++] = (x%10) + '0';
        x = x/10;
    }

    // If number of digits required is more, then
    // add 0s at the beginning
    while (i < d)
        str[i++] = '0';

    reverse(str, i);
    str[i] = '\0';
    return i;
}

// Converts a floating point number to string.
void ftoa(float n, char *res, int afterpoint)
```

```
{
    // Extract integer part
    int ipart = (int)n;

    // Extract floating part
    float fpart = n - (float)ipart;

    // convert integer part to string
    int i = intToStr(ipart, res, 0);

    // check for display option after point
    if (afterpoint != 0)
    {
        res[i] = '.';  // add dot

        // Get the value of fraction part upto given no.
        // of points after dot. The third parameter is needed
        // to handle cases like 233.007
        fpart = fpart * pow(10, afterpoint);

        intToStr((int)fpart, res + i + 1, afterpoint);
    }
}
```

DODATKOWE PROCEDURY

Wysyłanie komendy do LCD:

```
void LCD_writeCmd(uint8_t cmd) {
      LCD_RS_LOW;
      LCD_RW_LOW;
      LCD_writeByte(cmd);
}
```

Ustawienie pozycji kursora na ekranie LCD:

```
void LCD_cursorPosition(uint8_t x, uint8_t y) {
      switch(y) {
            case 0:
                  LCD_writeCmd(LCDC_SET_DDRAM | (LCD_LINE1 + x));
                  break;

            case 1:
                  LCD_writeCmd(LCDC_SET_DDRAM | (LCD_LINE2 + x));
                  break;
      }
}
```

Ustawienie linii danych jako wejście:

```
void LCD_pinsInput() {
        GPIO_InitTypeDef GPIO_InitStruct;
        GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}
```

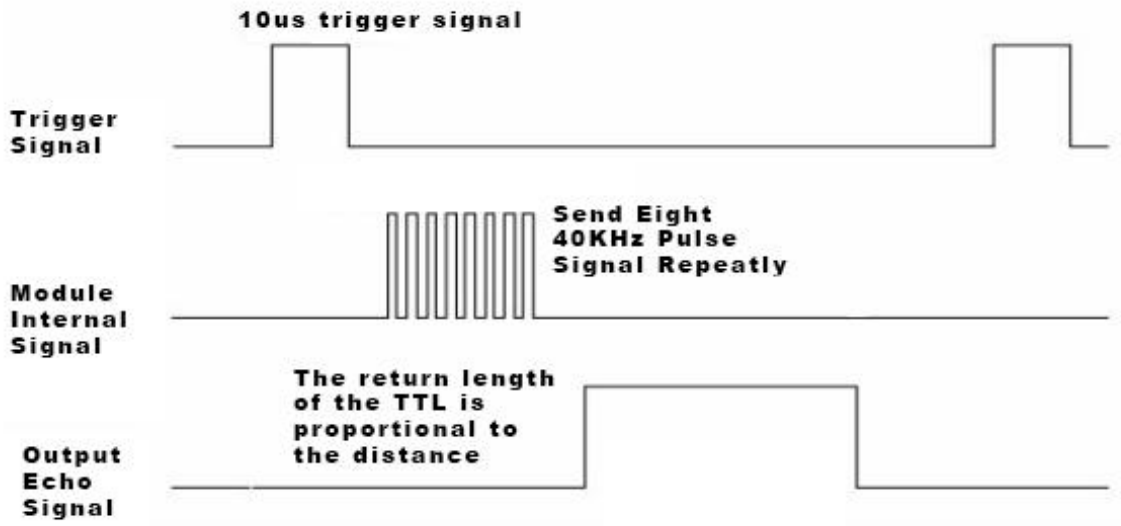
Ustawienie linii danych jak wyjście:

```
void LCD_pinsOutput() {
        GPIO_InitTypeDef GPIO_InitStruct;
        GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}
```

HC-SR04

SCHEMAT DZIAŁANIA

W celu wykonania pomiaru, linia *Trig* powinna otrzymać impuls (5V) przez minimum 10 [µs], to pozwoli na wygenerowanie 8 impulsów na częstotliwości 40 [kHz]. Kiedy czujnik wykryje wygenerowany ultradźwięk, ustawia linię *Echo* (5V) tak długo jak daleko znajduje się przeszkoda:



OBLICZANIE ODLEGŁOŚCI OD PRZESZKODY

Zmierzony czas ustawionej linii *Echo* podstawiamy do wzoru w celu obliczenia odległości.

Wzór:

$$L_{[m]} = \frac{T_{[s]} * 340\frac{m}{s}}{2}$$

Gdzie:

1. $\boldsymbol{L}$ – liczona odległość od przeszkody [m].
2. $\boldsymbol{T}$ – czas ustawionej linii *Echo* [s].
3. $\mathbf{340}\frac{\boldsymbol{m}}{\boldsymbol{s}}$ - prędkość rozchodzenia się dźwięku w powietrzu przy $\mathbf{15}$° [C].

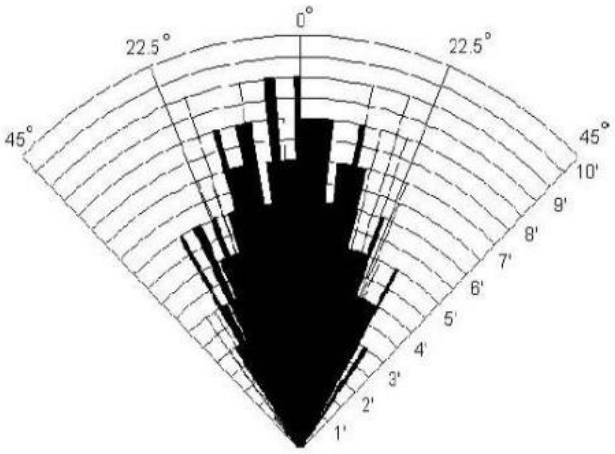
Można również przekształcić wzór i podzielić czas ustawionej linii *Echo* przez liczbę **58**:

$$L_{[cm]} = \frac{0{,}034\frac{cm}{\mu s} * T_{[\mu s]}}{2}$$

$$2L_{[cm]} = 0{,}034\frac{cm}{\mu s} * T_{[\mu s]}$$
$$T_{[\mu s]} = \frac{2}{0{,}034\frac{cm}{\mu s}} * L_{[cm]}$$
$$T_{[\mu s]} = 58 * L_{[cm]}$$
$$L_{[cm]} = \frac{T_{[\mu s]}}{58}$$

OGRANICZENIA I WARUNKI KORZYSTANIA

Według noty katalogowej produktu. Najlepiej przeprowadzać pomiary do $30°$ oraz na obiektach o powierzchni przynajmniej 0.5 [m^2]:



Ze względu na fakt, że prędkość dźwięku zależna jest od temperatury, ma to wpływ na przeprowadzane pomiary. Rodzaj przeszkody również odgrywa dużą rolę. Jeśli czujnik napotka delikatne materiały np. ubrania, wtedy odczyt będzie błędny.

KOD OBSŁUGI CZUJNIKA

W celu wykonania impulsu na linii *Trig*, wykorzystano *TIM2*, który przeładowuje się co 10 [µs]. W callback'u, który wykonuje się po przeładowaniu, następuje naprzemienna zmiana stanu logicznego linii *Trig*. Po zboczu opadającym następuje przerwa, która może być konfigurowalna przez użytkownika (ustawianie częstotliwości pomiarów). Domyślnie wynosi ona 200 [ms]:

```
int channel2 = 0;
volatile int trigger = 1;
int measure_frequency = 20000;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    // Impuls 10 [us] na trigger co 200 [ms]
      if (htim->Instance == TIM2) {
            if (trigger == 1) {
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
                  trigger = 0;
            } else if(trigger == 0) {
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);

                  static int trigger_gap = 0;
                  trigger_gap++;
```

```
                    if (trigger_gap >= measure_frequency) {
                          trigger_gap = 0;
                          trigger = 1;
                    }
              }
        }
}
```

Jeśli impuls został poprawnie przeprowadzony, następuje zliczenie czasu linii *Echo* w stanie wysokim. Do tego celu wykorzystano *TIM3* w trybie *PWM Input*, którego cykl wynosi 1 [µs] a okres 65536 [µs]. Posługując się callback'iem, odczytywane jest wypełnienie z *TIM_CHANNEL_2*, dzięki czemu będzie możliwe podstawienie wartości do wzoru na odległość:

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
      __disable_irq();

      HAL_TIM_Base_Stop_IT(&htim2);

      channel2 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_2) + 1;

      __HAL_TIM_SET_COUNTER(&htim2, 0);
      HAL_TIM_Base_Start_IT(&htim2);

      __enable_irq();
}
```

Kiedy zmierzony czas został pobrany, ustawiana jest flaga *trigger* na jeden co sprawia, że w pętli głównej wartość zostaje podstawiona do wzoru i wyświetlana na LCD:

```
while (1) {
  DecodeFrame();

  if(trigger) {
        char buff[20] = {0};
        float result = channel2/58.0;

        if(result < 10) {
              Buzz_on = 2;
              Buzz_off = 0;
        } else if((result >= 13) && (result <= 32)) {
              Buzz_on = 100;
              Buzz_off = 100;
        } else if((result >= 35) && (result <= 52)) {
              Buzz_on = 200;
              Buzz_off = 200;
        } else if((result >= 55) && (result <= 72)) {
              Buzz_on = 300;
              Buzz_off = 300;
        } else if((result >= 75) && (result <= 92)) {
              Buzz_on = 400;
              Buzz_off = 400;
        } else if((result >= 95) && (result <= 112)) {
              Buzz_on = 500;
              Buzz_off = 500;
        } else if((result >= 115) && (result <= 132)) {
              Buzz_on = 600;
```

```
            Buzz_off = 600;
        } else if(result > 135) {
            Buzz_on = 700;
            Buzz_off = 700;
        }

        ftoa(result, buff, 1);

        UART_Send_Tx("Distance: %s [cm]\r\n", buff);
        LCD_writeCmd(LCD_CLEAR);
        LCD_display(0, 0, "Distance:");
        LCD_display(0, 1, "%s [cm]", buff);
  }
}
```

BRZĘCZYK PIEZOELEKTRYCZNY

SCHEMAT DZIAŁANIA

Elementem generującym dźwięk jest membrana przytwierdzona do płytki wykonanej z piezoelektryka która zmienia swój kształt pod wpływem napięcia elektrycznego generując w ten sposób dźwięk. Zmieniając częstotliwość sygnału, zmieniamy ton wytwarzanego dźwięku. Zmieniając wypełnienie sygnału, zmieniamy głośność brzęczyka.

TON WYTWARZANEGO DŹWIĘKU

Brzęczyk wykorzystuje *TIM1* w trybie *PWM Generation CH1*. Aby obliczyć częstotliwość należy zastosować następujący wzór.

Wzór:

$$FREQ = \frac{TIM\_CLK}{(PSC + 1) * (ARR + 1)}$$

Gdzie:

1. $\boldsymbol{FREQ}$ – liczona częstotliwość.
2. $\boldsymbol{TIM\_CLK}$ – szybkość taktowania magistrali timer'a.
3. $\boldsymbol{PSC}$ – preskaler timer'a, który dzieli szybkość taktowania magistrali.
4. $\boldsymbol{ARR}$ – rejestr przechowujący wartość po której ma się przeładować timer.

W przypadku kiedy wiemy jaką chcemy zastosować częstotliwość, preskaler oraz znamy szybkość taktowania magistrali timer'a, możemy ze wzoru wyliczyć rejestr *ARR*.

Przykład:

$$2{,}5\ [kHz] = \frac{64\ [MHz]}{(63 + 1) * (ARR + 1)}$$

$$2500\ [Hz] = \frac{64000000\ [Hz]}{64 * (ARR + 1)}$$

$$160000\ [Hz] * (ARR + 1) = 64000000\ [Hz]$$

$$(ARR + 1) = \frac{64000000\ [Hz]}{160000\ [Hz]} = 400$$

GŁOŚNOŚĆ DŹWIĘKU

Sterując wypełnieniem, możemy określić głośność brzęczyka. Najniższy poziom to zero a najwyższy to połowa okresu timer'a. Zatem jeśli wartość rejestru *ARR* będzie równa 400, maksymalna głośność będzie określana jako 200.

CZĘSTOTLIWOŚĆ POMIARU

Do określania czasu włączenia i wyłączenia brzęczyka wykorzystano *SysTick*, który wywołuje callback co 1 [ms]:

```
volatile _Bool Buzz = 0;
volatile uint16_t Buzz_on = 0;
volatile uint16_t Buzz_off = 0;
int sound_level = 199;
int sound_frequency = 399;

void HAL_SYSTICK_Callback(void) {
	static uint32_t period = 0;
	period++;
	if((period >= 0) && (period <= Buzz_on-1))
		Buzz = 1;
	else if((period > Buzz_on-1) && (period <= Buzz_off+Buzz_on-1))
		Buzz = 0;
	else
		period = 0;
}
```

W odpowiednim momencie ustawia flagę, która sprawdzana jest w pętli głównej czy brzęczyk ma wydawać dźwięk:

```
if(Buzz)
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, sound_level);
else
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
```