POLITECHNIKA
BYDGOSKA
im. Jana i Jędrzeja Śniadeckich

POLITECHNIKA
BYDGOSKA
Wydział Telekomunikacji,
Informatyki i Elektrotechniki

.

MIKROPROCESORY

DOKUMENTACJA PROJEKTU

AMELIA KOWALCZYK
NAZWA FIRMY
SEMESTR III, GRUPA 1

SPIS TREŚCI

SPECYFIKACJA PROJEKTU

STM32 Nucleo-F303RE

1. Oprogramowanie komunikacji mikroprocesora z PC poprzez interfejs asynchroniczny z wykorzystaniem przerwań i buforów kołowych.

2. Zaprojektowanie i zaimplementowanie protokołu komunikacyjnego pozwalającego na:

- adresowanie ramek
- przekazywanie dowolnych danych
- weryfikację poprawności przesłanych danych z uwzględnieniem ich kolejności

3. Obsługa modemu komunikacji AT poprzez interfejs USATRT pracujący z DMA umożliwiająca wysyłanie SMS-ów oraz ich odbiór

PROTOKÓŁ KOMUNIKACJI

Komunikacja pomiędzy użytkownikiem, a modułem GSM SIM800L, będzie odbywać się za pomocą terminala.

W odpowiedzi na wysłane dane moduł będzie zwracał odpowiednie dane lub informacje o błędzie.

Komunikacja przez terminal z mikroprocesorem umożliwia:

- wysłanie SMS -a o dowolnej treści pod dowolny numer telefonu
- odczyt SMS-ów tylko przychodzących
- wyświetlenie SMS-a o określonym indeksie
- skasowanie SMS-a o określonym indeksie
- skasowanie wszystkich SMS-ów

OBSŁUGA BŁĘDÓW

Obsługa Błędów:

- Jeśli zostanie wprowadzona ramka z prawidłowym, adresem odbiorcy, a komenda nie zostanie rozpoznana, zostanie zwrócona informacja o błędzie.
- Jeśli moduł SIM nie będzie gotowy, ponawiam próby, nawiązania komunikacji.

- W przypadku braku znaku początku ramki / końca ramki, ramka zostanie odrzucona
- W przypadku niezgodnej sumy CRC ramka zostanie odrzucona
- Jeśli pojawi się znak ^, po którym nie występuje { } lub #, zostanie zwrócony błąd kodowania

RAMKA

Początek ramki	Nadawca	Odbiorca	Dane	Suma kontrolna	Koniec ramki
@	xxx	xxx	ASCI bez 0x00	CRC 8	;
0x40	ASCI – 3 bajty	ASCI – 3 bajty	Maksymalnie 176 bajty	2 – bajty ASCI ['0'-'9', 'A'-'F', 'a'-'f']	0x3B

Założenia ramki:

- Parametry liczbowe przesyłane są w formacie dziesiętnym jako ciąg znaków ASCII '0'-'9' (0x30 – 0x39), z wyjątkiem CRC, które jest przesyłane w formacie szesnastkowym jako ciąg znaków ASCII '0' – '9' i 'A'-'F' lub 'a'-'f'
- Maksymalna długość przesyłanych danych to 176 bajty
- Kodowanie:

Znak do przesłania	Znaki po kodowaniu
@	^{
;	^}
^	^#

SUMA KONTROLNA

Do sprawdzenia integralności przesłanych wykorzystuje algorytm CRC8 do obliczenia sumy kontrolnej. Przy jego wyliczaniu wykorzystuje wielomian $x^8 + 1$.

CRC8 opiera się na dzieleniu binarnym każdego bajtu danych przez wielomian, resztą tego dzielenia jest ostateczny wynik sumy CRC.

TABELA ROZKAZÓW

Dla wysyłania SMS -ów

W części ramki „dane” zawarta ma być komenda, numer telefonu, a po nim treść. Numer telefonu od komendy i treść od numeru telefonu należy oddzielić znakiem „:”

Dane [ASCI]		
Komenda	:Numer telefonu Dopuszczalny format: *(opcjonalnie) + <numer kierunkowy 2 liczby zgodnych z założeniami ramki >* (obowiązkowo) <numer telefonu 9 liczb zgodnych z założeniami ramki >	:Treść

Dla pozostałych komend

W części ramki „dane” wysyłana jest komenda i (opcjonalnie) jeden lub dwa parametry, poprzedzone znakiem „:”

Dane[ASCI]	
Komenda	[:Parametr[:Parametr1]]

Rozkaz składa się z komendy i przy niektórych komendach z parametru. W przypadku, niepodania parametru, kiedy jest to wymagane lub wprowadzeniu błędnie komendy zostanie zwrócony błąd.

Komendy:

Komenda	**Ilość bajtów**	**Znaczenie**
PI:<PIN> Od 4 – 8 liczb zgodnych z założeniami ramki	7-11	Wprowadzenie PINu do karty SIM
PU:<PUK>:<PIN> Puk 8 liczb zgodnych z założeniami ramki Pin Od 4 – 8 liczb zgodnych z założeniami ramki	13	Wprowadzenie PUK i ustawienie nowego PIN do karty SIM
RR	2	Odczyt wszystkich SMS-ów zapisanych w pamięci SIM
Odpowiedź: <indeks>,<status> , SMS from nr „< numer_telefonu_nadawcy_wiadomości >" received: <data> <treść wiadomości>		
RR:i	4	Odczyt SMS-ów przychodzących
Odpowiedź: <indeks>,<status> SMS from nr "<numer_telefonu_nadawcy_wiadomości >" received: <data> <treść wiadomości>		
RR:<indeks SMS-a>	Do 5	Wyświetlenie SMS-a o określonym indeksie
Odpowiedź: <status> SMS from nr "<numer_telefonu_nadawcy_wiadomości >" received: <data> <treść wiadomości> Po wyświetleniu SMS, zostanie wysłany dodatkowo ciąg znaków „OK”		

RM	2	Skasowanie wszystkich SMS-ów
Odpowiedź: All SMS have been deleted		
RM:<indeks SMS-a>	Do 5	Skasowanie SMS-a o określonym indeksie
Odpowiedź: SMS has been deleted		
RS:<numer telefonu>:<treść>	Do 176	Wysłanie SMS – a pod zadany numer
Odpowiedź: SMS sent successfully		

<Status>	**Znaczenie**
REC UNREAD	Wiadomość odebrana, ale nieprzeczytana
REC READ	Wiadomość odebrana i przeczytana

Komunikaty:

Komunikat	**Ilość bajtów**	**Znaczenie**
PIN required	12	PIN potrzebny
PUK & PIN required	18	Wymagane podanie PUK i nowego PINu
I am ready	10	Moduł GSM gotowy do działania

Wystąpienie błędu:

Komunikat	**Ilość bajtów**	**Znaczenie**
Invalid command	15	Nierozpoznany rozkaz
Data corruption	15	Niezgodna suma CRC
Wrong PIN	9	Błędny PIN
Wrong PUK	15	Błędny PUK
Recipient unknown	17	Nierozpoznany odbiorca
Connection failed	16	Błąd w nawiązaniu połączenia z modułem GSM
SMS too long	12	Próba wysłania zbyt długiego SMS

Encoding error	14	Błąd kodowania
Error	5	Wszystkie inne nieoczekiwane błędy
Buffer overflow	15	Bufor przepełniony

PARAMETRY KOMUNIKACJI

Konfiguracja UART1:

- Tryb: asynchroniczny
- Baund Rate: 38400 Bits/s
- Word length: 8 Bits
- Parity: None
- Stop Bits: wartość 1
- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples
- Piny:
  - PC4 – sygnał na pin – USART1_TX
  - PC5 – sygnał na pin – USART1_RX
- Przerwania: Aktywne

Konfiguracja UART2:

- Tryb: asynchroniczny
- Baund Rate: 38400 Bits/s
- Word length: 8 Bits
- Parity: None
- Stop Buts: wartość 1
- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples
- Piny:
  - PA2 – sygnał na pin – USART2_TX
  - PA3 – sygnał na pin – USART2_RX
- Przerwania: Aktywne

KOD I DZIAŁANIE

BUFOR KOŁOWY

Bufor kłowy cyklicznie nadpisuje stare – odczytane już dane, nowymi. Rozmiar buforów kołowych do odczytu i zapisu w tym projekcie wynoszą 4096 bajtów.

```
typedef struct {
	uint8_t* bufor;
	uint16_t start;
	uint16_t end;
} bufork_t;

extern bufork_t rxBufor2;
extern bufork_t txBufor2;

extern bufork_t rxBufor1;
extern bufork_t txBufor1;
```

Struktura danych bufor_t składa się z bufora (tablicy przechowującej dane), start-u - indeksu wskazującego na pierwszy wolny element, end-u indeksu wskazującego na pierwszy zajęty element (następny do odczytu). Instancje tej struktury:

- rxBufor2 – bufor kołowy do odbioru dla usart 2
- txBufor2 – bufor kołowy do transmisji dla usart 2
- rxBufor1 - bufor kołowy do odbioru dla usart 1
- txBufor1 - bufor kołowy do transmisji dla usart 1

OBSŁUGA BUFORA KOŁOWEGO

int8_t bufork_zapisz(bufork_t *buf, uint8_t dane) - funkcja odpowiedzialna za zapisanie znaku (określonego przez argument dane) do pierścieniowego bufora FIFO o zadanym rozmiarze.

- Jeśli bufor jest pełny (próba nadpisania danych), funkcja zwraca -1.
- W przypadku powodzenia zapisuje znak na aktualnej pozycji wskaźnika start, przesuwa wskaźnik do kolejnej pozycji i zwraca 0.

argumetry:

- bufork_t *buf – wskaźnik na strukturę opisującą bufor.
- uint8_t dane – bajt danych do zapisania w buforze.

Zwraca:

- 0 – zapis zakończony sukcesem.

- -1 – bufor jest pełny, zapis nie został wykonany.

```
int8_t bufork_zapisz(bufork_t *buf, uint8_t dane) {
	uint16_t tmp;
	tmp = (buf->start + 1) % BUF_LEN;
	if ( tmp == buf->end ) {
		return -1;
	} else {
		buf->bufor[buf->start] = dane;
		buf->start = tmp;
	}
	return 0;
}
```

int8_t bufork_odczyt(bufork_t *buf, uint8_t *data) - funkcja pozwalająca na odczytanie znaku bufora.

- Jeśli bufor jest pusty, funkcja zwraca -1.
- W przeciwnym wypadku odczytuje dane z aktualnej pozycji wskaźnika end, przesuwa wskaźnik do kolejnej pozycji i zwraca 0.

Parametry:

- bufork_t *buf – wskaźnik na strukturę opisującą bufor.
- uint8_t *data – wskaźnik, pod który zostanie zapisany odczytany bajt danych.

Zwraca:

- 0 – odczyt zakończony sukcesem.
- -1 – bufor jest pusty, odczyt nie został wykonany.

```
int8_t bufork_odczyt(bufork_t *buf, uint8_t *data)
{
	if (buf->start == buf->end)
		return -1;

	*data = buf->bufor[buf->end];
	buf->end++;
	if (buf->end == BUF_LEN) {
		buf->end = 0;
	}


	return 0;
}
```

KOMUNIKACJA

OBSŁUGA PRZERWAŃ

Przerwanie generowane przez przyjście znaku po UART-cie

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef*huart){
	if(huart->Instance== USART2){
		bufork_zapisz(&rxBufor2, znak);
		HAL_UART_Receive_IT(&huart2, &znak, 1);
	}

}
```

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) - funkcja obsługująca przerwanie zakończenia odbioru danych przez UART w trybie HAL_UART_Receive_IT.

- Jeśli przerwanie pochodzi z USART2:
  - Odczytany znak (znak) zostaje zapisany do bufora cyklicznego rxBufor2 za pomocą funkcji bufork_zapisz.
  - Następnie ponownie uruchamiany jest odbiór danych w trybie przerwaniowym (HAL_UART_Receive_IT), aby kontynuować odbieranie kolejnych znaków.

Parametry:

- UART_HandleTypeDef *huart – wskaźnik na strukturę opisującą interfejs UART, który wywołał przerwanie.

Przerwanie generowane przez wysłanie danych po UART-cie

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef*huart){
	if(huart->Instance== USART2){
		if (bufork_odczyt(&txBufor2, &znakTx) == 0) {
				HAL_UART_Transmit_IT(&huart2, &znakTx, 1);
		}
	}
}
```

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)- f unkcja obsługująca przerwanie zakończenia transmisji danych przez UART w trybie HAL_UART_Transmit_IT.

- Jeśli przerwanie pochodzi z USART2:
  - Próbuje odczytać kolejny znak z bufora cyklicznego txBufor2 za pomocą funkcji bufork_odczyt.
  - Jeśli odczyt zakończy się sukcesem, uruchamia transmisję kolejnego znaku w trybie przerwaniowym (HAL_UART_Transmit_IT).

Parametry:

- UART_HandleTypeDef *huart – wskaźnik na strukturę opisującą interfejs UART, który wywołał przerwanie.

Przerwanie od zdarzeń na UART-cie – przerwanie wywoływane jest przez UART1 po zakończeniu odbierania znaków i przejściu w stan IDLE

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
    if(huart->Instance == USART1)
```

```
    {
      for (uint8_t i = 0; i < Size; i++){
            if (bufork_zapisz(&rxBufor1, recBuff1[i]) == -1) {
                  sendf('Buffer overflow');
                  break;
            }
            recBuff1[i] = 0;
      }
      dma_transfer_complete == 0;
      HAL_UARTEx_ReceiveToIdle_DMA(&huart1, recBuff1, 4096);
    }
}
```

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)- funkcja obsługująca przerwanie zakończenia transmisji danych przez UART w trybie HAL_UART_Transmit_IT.

- Jeśli przerwanie pochodzi z USART1:
  - Próbuje odczytać kolejny znak z bufora cyklicznego txBufor1 za pomocą funkcji bufork_odczyt.
  - Jeśli odczyt zakończy się sukcesem, uruchamia transmisję kolejnego znaku w trybie przerwaniowym (HAL_UART_Transmit_IT).

Parametry:

- UART_HandleTypeDef *huart – wskaźnik na strukturę opisującą interfejs UART, który wywołał przerwanie.

KOMUNIKACJA Z MODUŁEM GSM

Odczyt znaków, które pojawiły się na UART1

```
void odczytSIM(uint8_t* pom){
      uint16_t i = 0;

      while (rxBufor1.start != rxBufor1.end && isspace(rxBufor1.bufor[rxBufor1.end])){
            bufork_odczyt(&rxBufor1, &pom[0]);
      }
      while (bufork_odczyt(&rxBufor1, &pom[i++]) == 0);
      i = strlen(pom) - 1;
    while (isspace(pom[i])) {
      pom[i--] = 0;
    }

}
```

void odczytSIM(uint8_t *pom) - funkcja odpowiedzialna za odczyt danych z bufora cyklicznego rxBufor1 i zapis ich do podanej tablicy pom. Dane są odczytywane, a zbędne białe znaki (na początku i końcu) są usuwane.

Działanie funkcji:

1. Pominięcie początkowych białych znaków:

- Funkcja najpierw sprawdza, czy dane na początku bufora rxBufor1 są białymi znakami (np. spacje, tabulatory, nowe linie).
- Jeśli tak, dane są ignorowane, a wskaźnik w buforze przesuwany za pomocą bufork_odczyt.

2. Odczyt danych do tablicy:
   - Kolejne dane są odczytywane z bufora rxBufor1 i zapisywane do tablicy pom aż do momentu, gdy bufor jest pusty.
3. Usunięcie końcowych białych znaków:
   - Po zakończeniu odczytu funkcja sprawdza białe znaki na końcu tablicy pom (korzystając z funkcji strlen do wyznaczenia długości danych).
   - Białe znaki są zamieniane na 0, aby wyczyścić końcówkę danych.

Parametry:

- uint8_t *pom – wskaźnik na tablicę, w której mają zostać zapisane odczytane dane (z bufora rxBufor1) bez białych znaków na początku i końcu.

WYSYŁANIE PO UART2

```
void sendf(char* message,...) {
	char tmp[BUF_LEN];
	va_list arguments;
	va_start(arguments, message);
	vsprintf(tmp, message, arguments
	va_end(arguments);
	for (uint16_t i =0; i < strlen(tmp); i++) {
		bufork_zapisz(&tmp_txBuf, tmp[i]);
	}


	__disable_irq
	if ((txBufor2.start==txBufor2.end)&&(__HAL_UART_GET_FLAG(&huart2,
UART_FLAG_TXE)==SET)) {
		txBufor2.start = tmp_txBuf.start;
		uint8_t ztmp;
		bufork_odczyt(&txBufor2, &ztmp);
		HAL_UART_Transmit_IT(&huart2, &ztmp, 1);
	} else {
		txBufor2.start = tmp_txBuf.start;
	}

	__enable_irq();
}
```

void sendf(char *message, ...) - funkcja odpowiedzialna za formatowanie i wysyłanie danych tekstowych przez UART2 w trybie przerwaniowym.

Pozwala na przesłanie sformatowanego ciągu znaków (w stylu printf) do kolejki transmisji UART2, jednocześnie zarządzając buforami cyklicznymi.

OBSŁUGA RAMKI

OBLICZENIE SUMY CRC

Algorytm sumy CRC umożliwia kontrolę poprawności odebranych danych.

```
#define POLYNOMIAL 0x81
#define CRC_INITIAL 0x00
uint8_t oblicz_crc(const uint8_t *buf, uint8_t len){
       uint8_t crc = CRC_INITIAL;
          for (uint8_t i = 0; i < len; i++) {
              crc ^= buf[i];
              for (uint8_t bit = 0; bit < 8; bit++) {
                  if (crc & 0x80) {
                      crc = (crc << 1) ^ POLYNOMIAL;
                  } else {
                      crc <<= 1;
                  }
              }
          }
          return crc;
}
```

#define POLYNOMIAL 0x81 - stała definiująca wielomian wykorzystywany w algorytmie obliczania CRC Jest to wartość używana w operacjach XOR podczas obliczeń.
#define CRC_INITIAL 0x00 - stała definiująca początkową wartość rejestru CRC. To wartość, od której zaczynają się obliczenia.

uint8_t oblicz_crc(const uint8_t *buf, uint8_t len) - funkcja oblicza 8-bitową sumę kontrolną CRC dla podanego bufora danych na podstawie algorytmu CRC-8 z użyciem zdefiniowanego wielomianu POLYNOMIAL.
Wartość CRC jest obliczana w sposób iteracyjny, przechodząc przez wszystkie bajty danych oraz każdy bit w bajcie.

Parametry:

- const uint8_t *buf – wskaźnik na bufor danych wejściowych, dla których ma zostać obliczona suma kontrolna.
- uint8_t len – długość bufora danych (liczba bajtów).

Zwraca:

- uint8_t – obliczona wartość CRC (8-bitowa suma kontrolna).

Działanie funkcji:

1. Inicjalizuje zmienną crc wartością CRC_INITIAL.
2. Iteruje przez wszystkie bajty w buforze.
3. Dla każdego bajtu wykonuje operację XOR z bieżącą wartością CRC.
4. Dla każdego bitu w bieżącym bajcie sprawdza najstarszy bit (crc & 0x81):
   - Jeśli jest ustawiony, wykonuje przesunięcie w lewo i operację XOR z POLYNOMIAL.
   - W przeciwnym przypadku przesuwa crc w lewo.
5. Zwraca obliczoną wartość CRC.

FUNKCJE DO ODBIORU I DEKODOWANIA RAMKI

```
uint8_t dopisz_znak_ramki(uint8_t* ramka) {
	if(rxBufor2.start==rxBufor2.end){
		return 2;
	}
	uint8_t znak;
	uint8_t len_r = strlen(ramka);
	bufork_odczyt(&rxBufor2, &znak);
	if (len_r == 0 && znak == '@') {
		ramka[0] = znak;
		ramka[1] = 0;
	}else if(len_r>0){
		if (len_r==1 && ramka[0] == '@') {
			len_r = 0;
		}
		if (znak == '@') {
			ramka[0] = '@';
			ramka[1] = 0;
		} else if (znak == ';') {
			uint8_t crc[5]="0x  ";
			uint8_t obliczone_crc, przeslane_crc, i;
			uint8_t nadawca[3];
			uint8_t odbiorca[3];
			uint8_t komenda[200];
			crc[2]=ramka[len_r-2];
			crc[3]=ramka[len_r-1];
			if(czy_HEX_ok(&crc)!=0){
				sendf("Data corruption\n");
				return 2;
			}
			ramka[len_r-2]=0;
			przeslane_crc = strtol(&crc,NULL,0);
			obliczone_crc = oblicz_crc(ramka, strlen(ramka));
			if (obliczone_crc != przeslane_crc) {
				sendf("Data corruption\n");
				ramka[0] = 0;
				return 2;
			  } else {
				wybierz(ramka, nadawca, 3);
				wybierz(ramka+3, odbiorca, 3);
				if(/*strcmp(nadawca, "TER")== 0 &&*/ strcmp(odbiorca,
"GSM")!= 0){
```

```
                                sendf("Recipient unknown\n");
                                ramka[0] = 0;
                                return 2;
                        } else {
                                return 0;
                        }

                }
        } else {
                if(ramka[len_r-1]=='^') {
                        if (znak=='{') {
                                ramka[len_r-1] = '@';
                                ramka[len_r] = 0;
                        } else if(znak=='}'){
                                ramka[len_r-1] = ';';
                                ramka[len_r] = 0;
                        } else if(znak=='#'){
                                ramka[len_r-1] = '^';
                                ramka[len_r] = 0;
                        }else{
                                sendf("Encoding error\n");
                                return 2;
                        }
                } else {
                        ramka[len_r] = znak;
                        ramka[len_r+1] = 0;
                }
        }

    }
    return 1;

}
```

uint8_t dopisz_znak_ramki(uint8_t *ramka) - funkcja odpowiedzialna za obsługę znaków ramki danych w systemie komunikacji. Jej zadaniem jest dodawanie kolejnych znaków do bufora ramki (ramka) na podstawie danych odczytanych z bufora kołowego rxBufor2. Funkcja rozpoznaje początek (@), koniec (;) oraz specjalne znaki w ramce, a także weryfikuje integralność danych za pomocą CRC.

Działanie funkcji:

1. Sprawdzenie dostępności danych:
    - Jeśli bufor rxBufor2 jest pusty (brak znaku do odczytania), funkcja zwraca wartość 2.
2. Odczyt znaku z bufora:
    - Funkcja odczytuje kolejny znak z bufora rxBufor2 i sprawdza długość aktualnej ramki (len_r).
3. Rozpoznanie początku ramki:
    - Jeśli ramka jest pusta (len_r == 0), funkcja akceptuje tylko znak początku ramki (@). Inne znaki są ignorowane.
4. Obsługa znaków wewnątrz ramki:

- Jeśli odczytany znak to @, a ramka nie została zakończona (; nie wystąpiło), funkcja resetuje ramkę, rozpoczynając nową sekwencję od znaku @.

5. Rozpoznanie końca ramki:
   - Jeśli odczytany znak to ;, funkcja:
     - Weryfikuje poprawność przesłanego CRC na podstawie obliczeń funkcji oblicz_crc.
     - Sprawdza poprawność struktury ramki (adresy nadawcy i odbiorcy) i ewentualnie raportuje błędy.
6. Dekodowanie znaków specjalnych:
   - Dekoduje znaki zgodnie z tabelą komunikacji
   - Błędne kodowanie jest sygnalizowane za pomocą komunikatu Encoding error.
7. Zakończenie przetwarzania:
   - Jeśli ramka jest zakończona poprawnie (; i poprawne CRC), funkcja zwraca 0.
   - W przypadku problemów (błędy CRC, nieznany odbiorca, błędne kodowanie) funkcja zwraca 2.
   - Jeśli ramka jest w trakcie budowy, zwracana jest wartość 1.

Parametry:

- uint8_t *ramka – wskaźnik na bufor, w którym przechowywane są znaki ramki.

Zwracane wartości:

- 0 – ramka została poprawnie zakończona i przetworzona.
- 1 – ramka jest w trakcie budowy (dodano kolejny znak).
- 2 – wystąpił błąd (brak danych w buforze, błąd CRC, błędne kodowanie lub nieznany odbiorca).

OBSŁUGA KOMEND

```
void obsluga_komend(uint8_t* komenda){
      uint8_t ramka[BUF_LEN];
      if(strlen((char*)komenda)==2){

/**********Odczyt wszystkich sms zapisanych w pamięci*/
```

```
            if(strcmp((char*)komenda, "RR")== 0){
                  sendSIM("AT+CMGL=\"ALL\"\r");
            }else if(strcmp(komenda, "RM")== 0){
                  sendSIM("AT+CMGDA=\"DEL ALL\"\r");
            }
      }else if(komenda[2]/**(komenda+2)*/==':'){
            uint8_t rozkaz[12];
            wybierz(komenda,&rozkaz,2);

/**********************************wysyłanie sms*/
            if(strcmp(rozkaz, "RS")== 0){
                  char tmp[200];
                  char odp[100];
                  uint8_t tel[13];

                  if(komenda[3]=='+'){
                        if(komenda[15]==':'){
                              wybierz(komenda+3, &tel,12);
                              wybierz (komenda + 16 ,&tresc,strlen(komenda)-16 );

                              }
                        }else{
                              if(komenda[12]==':'){
                                    tel[0]='+';
                                    tel[1]='4';
                                    tel[2]='8';
                                    wybierz(komenda+3,&tel[3],9);
                                    wybierz (komenda + 13 ,tresc,strlen(komenda)-13 );
                              }
                        }
                        if(czy_liczba(&tel[1])==1){
                        sendf("Error\n");
                        return;
                  }
                        if (strlen(tresc) > 0)
                        sprintf(&tmp, "AT+CMGS=\"%s\"\r", tel);
                        sendSIM(&tmp);
            }else{
                  uint8_t parametr[5];
                  char tmp[20];
                  wybierz(komenda + 3, &parametr,strlen(komenda)-3 );
                  if(strcmp(rozkaz, "RR")==0){

/*****************************Odczyt sms przychodzacych*/
                        if(strcmp(parametr, "i")==0){
                              sendSIM("AT+CMGL=\"REC UNREAD\"\n");
                              sendSIM("AT+CMGL=\"REC READ\"\n");
/**************************Odczyt SMS o zadanym indeksie*/
                        }else{
                              if(czy_liczba(&parametr)==1){
                                    sendf("Invalid command\n");
                                    return;
                              }
                              sprintf(&tmp, "AT+CMGR=%s\r",&parametr);
                              sendSIM(&tmp);
                        }
/************************Usuniecie SMS o zadanym indeksie*/
                  }else if(strcmp(rozkaz, "RM")==0){
```

```
                sprintf(&tmp, "AT+CMGD=%s\n",&parametr);
                sendSIM(&tmp);

        }else{
                sendf("Invalid command\n");
                }
        }
}

}
```

void obsluga_komend(uint8_t *komenda)- funkcja odpowiedzialna za analizę i obsługę komend odebranych z ramki. Na podstawie zawartości komenda, wykonuje różne akcje, takie jak odczyt wiadomości SMS, ich usuwanie czy wysyłanie nowej wiadomości.

Działanie funkcji:

1. Sprawdzenie długości komendy:
    - Funkcja najpierw sprawdza, czy komenda ma długość równą 2 znakom. Jeśli tak, obsługuje proste komendy:
        - RR: Wyświetlenie wszystkich zapisanych wiadomości SMS.
        - RM: Usunięcie wszystkich wiadomości.
2. Obsługa komend z dodatkowymi parametrami:
    - Jeśli trzeci znak w komendzie to ':', funkcja rozpoznaje bardziej złożone komendy:
        - Wyodrębnia polecenie (pierwsze dwa znaki) i parametry (znaki po ':').
        - Obsługuje różne przypadki na podstawie rozpoznanej komendy:
3. Wysyłanie wiadomości SMS (RS):
    - Rozpoznaje numer telefonu i treść wiadomości na podstawie parametrów.
    - Waliduje numer telefonu (czy składa się wyłącznie z cyfr).
    - Przygotowuje polecenie AT dla wysyłania wiadomości SMS i przekazuje je do modułu GSM.
4. Odczyt wiadomości SMS (RR):
    - Jeśli parametr to i, wyświetla wiadomości przychodzące (nowe i przeczytane).
    - Jeśli parametr jest liczbą, wyświetla wiadomość SMS o zadanym indeksie.
5. Usuwanie wiadomości SMS (RM):

- Usuwa wiadomość SMS o zadanym indeksie.

6. Obsługa błędów:
   - W przypadku nierozpoznanej komendy lub błędnych parametrów, wysyłany jest komunikat o błędzie (Invalid command lub Error).

Parametry:

- uint8_t *komenda – wskaźnik na bufor zawierający komendę odebraną z ramki.

INICJALIZACJA

INICJALIZACJA

```
void initSIM800() {
    sendMA("AT");
    char pom[10];
      while (rxBufor1.start == rxBufor1.end);
    odczytSIM(pom);
    uint32_t tp = HAL_GetTick();
    while (strstr(pom, "OK") == NULL) {
        sendMA("AT");
      while (rxBufor1.start == rxBufor1.end);
        odczytSIM(pom);
        if(HAL_GetTick()-tp <30000){
            HAL_UART_Transmit(&huart2, "Connection failed\n", 18, 1500);
            HAL_NVIC_SystemReset();
        }
    }
    sendMA("ATE0");
      while (rxBufor1.start == rxBufor1.end);
    odczytSIM(pom);
    sendMA("AT+CMGF=1");
      while (rxBufor1.start == rxBufor1.end);
    odczytSIM(pom);
    uint8_t buff1[40];
    uint8_t buff2[20];
    sendMA("AT+CPIN?");
      while (rxBufor1.start == rxBufor1.end);
    odczytSIM(&buff1);
    while(strncmp(&buff1, "+CPIN: READY", 12)!= 0){
      uint8_t puk[9];
      uint8_t pukpin[18];
      uint8_t pin[9];

      if(strcmp(&buff1, "+CPIN: SIM PUK")== 0){
            sendf("PUK & PIN required\n");
            if (dekoduj_ramke(&pukpin)!=0){
                  continue;
          }
            if(pukpin[11]!=':' || pukpin[2]!=':'){
                  sendf("Error");
                continue;
            }
```

```
            wybierz(&pukpin, &puk, 2);
            if(strcmp(pukpin, "PU")!= 0){
                  sendf("Invalid command\n");
                  continue;
            }
            wybierz(&pukpin+3, &puk, 8);
            wybierz(&pukpin+12, &pin, strlen(&pukpin)-12);
            if(czy_liczba(&puk)==1){
                  sendf("Wrong PUK\n");
            }
            if(czy_liczba(&pin)==1){
                  sendf("Wrong PIN\n");
            }
            sprintf(&buff2, "AT+CPIN=\"%s\",\"%s\"",&puk,&pin);
            sendMA(&buff2);
            while (rxBufor1.start == rxBufor1.end);
            odczytSIM(&buff1);
      }else if(strcmp(&buff1, "+CPIN: SIM PIN")== 0||(strcmp(&buff1, "ERROR")== 0)){
            sendf("PIN required\n");
            if (dekoduj ramke(&pukpin)!=0){
                  continue;
            }
            if(pukpin[2]!=':'){
                  sendf("Error");
                  continue;
            }
          wybierz(&pukpin, &pin, 2);
          if(strcmp(pukpin, "PI")!= 0){
            sendf("Invalid command\n");
              continue;
          }
          wybierz(&pukpin+3, &pin, strlen(&pukpin)-3);
            if(czy_liczba(&pin)==1){
                  sendf("Wrong PIN\n");
            }
          sprintf(&buff2, "AT+CPIN=\"%s\"",&pin);
                  sendMA(&buff2);
                  while (rxBufor1.start == rxBufor1.end);
          odczytSIM(&buff1);
      }
      sendMA("AT+CPIN?");
      while (rxBufor1.start == rxBufor1.end);
      odczytSIM(&buff1);
    }

    sendMA("AT+CNMI=1,1,0,0,0");
      while (rxBufor1.start == rxBufor1.end);
      odczytSIM(&buff1);

}
```

void initSIM800()- funkcja inicjalizująca moduł SIM800, odpowiedzialna za nawiązanie komunikacji, sprawdzenie statusu karty SIM oraz konfigurację modułu GSM do obsługi wiadomości SMS.

Działanie funkcji:

1. Inicjalizacja komunikacji z modułem SIM800:
    - Funkcja wysyła polecenie AT w celu sprawdzenia czy moduł SIM800 odpowiada.
    - W pętli sprawdzana jest odpowiedź na wysłane polecenie. Jeśli moduł nie odpowie w ciągu 30 sekund, następuje reset mikrokontrolera.
2. Konfiguracja modułu:
    - Wyłączenie echa za pomocą polecenia ATE0 (moduł nie będzie zwracał wysyłanych komend).
    - Ustawienie trybu tekstowego dla SMS-ów za pomocą AT+CMGF=1.
3. Sprawdzenie statusu karty SIM:
    - Polecenie AT+CPIN? jest wysyłane w celu uzyskania statusu karty SIM.
    - Analiza odpowiedzi na +CPIN:
        - +CPIN: READY: Karta SIM jest gotowa do pracy.
        - +CPIN: SIM PIN: Wymagane jest wprowadzenie kodu PIN.
        - +CPIN: SIM PUK: Wymagane jest wprowadzenie kodu PUK oraz nowego PIN-u.
4. Obsługa kodów PIN i PUK:
    - Jeśli wymagany jest kod PIN lub PUK, funkcja czeka na odpowiednie dane (ramka z kodami PIN/PUK) i weryfikuje je:
        - Kod PIN/PUK jest sprawdzany pod kątem poprawności (czy zawiera tylko cyfry).
        - Po wprowadzeniu poprawnego kodu, wysyłane jest odpowiednie polecenie AT+CPIN="PIN" lub AT+CPIN="PUK","PIN".
    - Jeśli dane są błędne, wysyłany jest komunikat o błędzie i ponawiana jest próba.
5. Konfiguracja obsługi SMS:
    - Wysyłane jest polecenie AT+CNMI=1,1,0,0,0, które konfiguruje moduł, aby informował o nowych wiadomościach SMS zapisanych na karcie SIM (UART zwraca: +CMTI: "SM",<index>).

FUNKCJA POMOCNICZA – DO OBSŁUGI BUFORU POZA PĘTLĄ GŁÓWNĄ

```
uint8_t dekoduj_ramke( uint8_t* komenda ){
	uint8_t ramka[BUF_LEN];
	uint8_t pom;


	while (rxBufor2.start != rxBufor2.end){
		pom = dopisz_znak_ramki(&ramka);
	}
	if (pom== 0) {
		// Odczytano prawidłowo ramkę
		wybierz(&ramka[6],komenda ,strlen(ramka)-6);
		return 0;
	} else {
		// ramka nieprawidłowa
		sendf('Error data');
		return 1;
	}
}
```

uint8_t dekoduj_ramke(uint8_t *komenda)- funkcja odpowiedzialna za odczyt i dekodowanie pełnej ramki danych z bufora rxBufor2. Używa funkcji dopisz_znak_ramki do budowy ramki oraz wyodrębnienia komendy z jej zawartości. W przypadku wystąpienia błędu w ramce funkcja raportuje problem i zwraca odpowiedni kod statusu. Wykorzystywana jest tylko w momencie inicjalizacji pracy z modułem, przed pętlą główną.

Działanie funkcji:

1. Inicjalizacja:
    - Bufor ramka o maksymalnym rozmiarze BUF_LEN jest przygotowywany na dane ramki.
2. Budowanie ramki:
    - W pętli odczytywane są znaki z bufora rxBufor2 za pomocą funkcji dopisz_znak_ramki, aż bufor zostanie opróżniony.
3. Sprawdzenie poprawności ramki:
    - Jeśli funkcja dopisz_znak_ramki zwróci 0, oznacza to poprawnie zakończoną ramkę.
    - Komenda (zawartość ramki poza nagłówkiem i CRC) zostaje wyodrębniona z użyciem funkcji wybierz, począwszy od siódmego bajtu ramki.
4. Obsługa błędów:
    - W przypadku błędnej ramki (wartość inna niż 0 zwrócona przez dopisz_znak_ramki) funkcja wysyła komunikat o błędzie (Error data) i zwraca kod błędu.

Parametry:

- uint8_t *komenda – wskaźnik na bufor, do którego zostanie skopiowana wyodrębniona komenda z ramki.

Zwracane wartości:

- 0 – ramka została poprawnie odczytana i komenda została wyodrębniona.
- 1 – wystąpił błąd podczas dekodowania ramki (np. błędne dane, brak zakończenia ramki).

WYSYŁANIE PO DMA - USART1

```
void sendMA(char* message){
	uint8_t pom[1024];
	HAL_StatusTypeDef stat;

	sprintf((char *)&pom, "%s\r\n", message);
	stat = HAL_UART_Transmit_DMA(&huart1,&pom,strlen(pom));
}
```

void sendMA(char *message)- funkcja odpowiedzialna za wysłanie ciągu znaków przez UART1 w trybie DMA. Funkcja automatycznie dodaje znaki zakończenia linii (\r\n) do wysyłanej wiadomości. Wykorzystywana jest tylko w trakcie inicjalizacji.

Działanie funkcji:

1. Formatowanie wiadomości:
    - Funkcja kopiuje zawartość argumentu message do lokalnej tablicy pom (o rozmiarze 1024 bajtów), jednocześnie dodając na końcu znaki nowej linii (\r\n) przy użyciu sprintf.
2. Rozpoczęcie transmisji DMA:
    - Sformatowany ciąg znaków znajdujący się w buforze pom jest przesyłany przez UART1 za pomocą funkcji HAL_UART_Transmit_DMA.
    - Transmisja w trybie DMA pozwala na asynchroniczne przesyłanie danych bez blokowania procesora.

Parametry:

- char *message – wskaźnik na ciąg znaków, który ma być wysłany przez UART.

Działanie funkcji:

1. Formatowanie wiadomości:

- Funkcja przyjmuje sformatowany ciąg znaków (message) z dowolną liczbą dodatkowych argumentów.
- Używając funkcji vsprintf, przygotowuje sformatowaną wiadomość i zapisuje ją w tymczasowym buforze tmp.

2. Dodanie do bufora transmisyjnego:
   - Każdy znak z tablicy tmp jest zapisywany do tymczasowego bufora cyklicznego tmp_txBuf za pomocą bufork_zapisz.
3. Rozpoczęcie transmisji UART:
   - Wyłączane są przerwania (__disable_irq) w celu zabezpieczenia przed kolizją danych.
   - Jeśli bufor txBufor2 jest pusty oraz rejestr transmisji UART (flaga UART_FLAG_TXE) jest gotowy, to:
     - Dane z tymczasowego bufora są przenoszone do właściwego bufora txBufor2.
     - Rozpoczynana jest transmisja pierwszego znaku za pomocą funkcji HAL_UART_Transmit_IT.
   - W przeciwnym wypadku dane z tmp_txBuf są przenoszone do txBufor2 bez natychmiastowego rozpoczęcia transmisji.
4. Wznowienie przerwań:
   - Po zakończeniu aktualizacji buforów, przerwania są włączane (__enable_irq).

PĘTLA GŁÓWNA

```
while (1)
  {
      if(rxBufor2.start!=rxBufor2.end && dma_transfer_complete == 1){
              if (dopisz_znak_ramki(&ramka)==0) {
                    uint8_t komenda[200];
                    wybierz(&ramka[6],&komenda ,strlen(ramka)-6);
                    obsluga_komend(&komenda);
                    ramka[0] = 0;

              }
        }

        if (rxBufor1.start != rxBufor1.end) {
              uint8_t bufor[1024];
              odczytSIM(&bufor);
              if (strncmp(&bufor, "+CMGR:", 6) == 0 || strncmp(&bufor, "+CMGL:", 6) ==
0) {
```

```
            char * w = strtok(&bufor, ",");
                  sendf("%s, ", w+7);
                  if (strncmp(&bufor, "+CMGL:", 6) == 0) {
                        w = strtok(NULL, ",");
                        sendf("%s, ", w);
                  }
                  w = strtok(NULL, ",");
                  sendf("SMS from nr %s\n", w);
                  strtok(NULL, ",");
                  w = strtok(NULL, "\r\n");
                  sendf("received: %s", w);
                  w = strtok(NULL, "");
                  sendf("%s\r\n----------------------\r\n", w);
            } else if (bufor[0] == '>') {
                  if (strlen(tresc) > 0) {
                        uint8_t tmp[200];
                        sprintf(&tmp, "%s\x1A", tresc);
                        sendSIM(&tmp);
                        tresc[0] = 0;
                  }
            } else if (strncmp(&bufor, "+CMGS:", 6)==0) {
                  if (strstr(&bufor, "OK") != NULL) {
                        sendf("SMS sent successfully");
                  } else {
                        sendf("Error sent SMS");
                  }

            }
            dma_transfer_complete = 1;
      }

      if (txBufor1.start != txBufor1.end && dma_transfer_complete == 1) {
            uint8_t pom[200],i=0;
            while (bufork_odczyt(&txBufor1, &pom[i++])==0);
            pom[--i] = 0;
            dma_transfer_complete = 0;
            HAL_UART_Transmit_DMA(&huart1,&pom,strlen(pom));
      }


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Pętla główna realizuje obsługę komunikacji z modułem SIM800L oraz zarządzanie przesyłanymi danymi. Składa się z trzech głównych bloków odpowiedzialnych za: przetwarzanie odebranych danych, obsługę poleceń SMS, a także przesyłanie danych do modułu SIM.

1. Obsługa odebranych danych przez DMA (rxBufor2):

   - Sprawdzane jest, czy bufor rxBufor2 zawiera nowe dane (rxBufor2.start != rxBufor2.end) oraz czy transmisja DMA została zakończona (dma_transfer_complete == 1).

   - Odebrane dane są dołączane do tablicy ramka za pomocą funkcji dopisz_znak_ramki().

- Jeśli ramka jest kompletna (dopisz_znak_ramki() zwraca 0):
  - Komenda jest wydzielana z ramki za pomocą funkcji wybierz() i przekazywana do wykonania funkcji obsluga_komend().
  - Po przetworzeniu polecenia tablica ramka zostaje wyzerowana.

2. Obsługa danych przychodzących z modułu SIM (rxBufor1):
   - Jeśli rxBufor1 zawiera nowe dane (rxBufor1.start != rxBufor1.end), funkcja odczytSIM() odczytuje je i przekazuje do dalszego przetwarzania.
   - Odczytane dane są analizowane:
     - +CMGR: lub +CMGL::
       - Jeśli odebrano wiadomość SMS lub listę wiadomości, dane są parsowane i wyświetlane (numer nadawcy, data, czas, treść).
     - >:
       - Moduł oczekuje treści wiadomości SMS do wysłania. Funkcja sprawdza, czy zmienna tresc zawiera treść, a następnie wysyła SMS (zakończony znakiem \x1A).
     - +CMGS::
       - Potwierdzenie wysłania wiadomości. Jeśli odpowiedź zawiera OK, SMS został wysłany pomyślnie; w przeciwnym razie zwracany jest błąd.
   - Po zakończeniu operacji flaga dma_transfer_complete jest ustawiana na 1.
3. Wysyłanie danych do modułu SIM (txBufor1):
   - Jeśli txBufor1 zawiera dane do wysłania (txBufor1.start != txBufor1.end) i transmisja DMA jest zakończona (dma_transfer_complete == 1):
     - Dane są odczytywane z txBufor1 za pomocą funkcji bufork_odczyt().
     - Po przygotowaniu danych transmisja DMA zostaje zainicjowana za pomocą funkcji HAL_UART_Transmit_DMA().

INNE FUNKCJE – FUNKCJE WALIDUJĄCE, DZIAŁAJĄCE NA STRINGACH

FUNKCJE WALIDACYJNE

```
uint8_t czy_liczba(uint8_t * liczba){
	for(uint8_t i =0; i< strlen(liczba); i++){
		if (isdigit(liczba[i]) == 0) {
			return 1;
		}
	}
	return 0;

}
```

uint8_t czy_liczba(uint8_t * liczba) - funkcja sprawdza, czy podany ciąg znaków (liczba) składa się wyłącznie z cyfr dziesiętnych
Działanie funkcji:

1. Iteracja po znakach:
   - Funkcja przechodzi przez każdy znak w podanym ciągu liczba.
   - Długość ciągu określana jest za pomocą funkcji strlen().
2. Sprawdzanie cyfr:
   - Każdy znak w ciągu jest sprawdzany za pomocą funkcji isdigit(), która weryfikuje, czy znak jest cyfrą (0–9).
   - Jeśli którykolwiek znak w ciągu nie jest cyfrą, funkcja zwraca wartość 1.
3. Zwracanie wyniku:
   - Jeśli wszystkie znaki w ciągu są cyframi, funkcja zwraca 0, co oznacza, że ciąg jest poprawną liczbą dziesiętną.

Parametry:

- uint8_t * liczba: wskaźnik na ciąg znaków do sprawdzenia.

Zwracane wartości:

- 0: ciąg składa się wyłącznie z cyfr dziesiętnych.
- 1: ciąg zawiera co najmniej jeden znak, który nie jest cyfrą.

```
uint8_t czy_HEX_ok(uint8_t * liczba){
	uint8_t ok = 2;
	if((liczba[2]>='A'&&liczba[2]<='F')||(liczba[2]>='a'&&liczba[2]<='f')||(liczba[2]
>='0'&&liczba[2]<='9')){
		ok --;
	}
	if((liczba[3]>='A'&&liczba[3]<='F')||(liczba[3]>='a'&&liczba[3]<='f')||(liczba[3]
>='0'&&liczba[3]<='9')){
		ok --;
	}
	return ok;
}
```

uint8_t czy_HEX_ok(uint8_t * liczba) -funkcja weryfikuje, czy podany ciąg znaków (liczba) zawiera poprawne znaki w dwóch ostatnich pozycjach, które są zgodne z zapisem szesnastkowym (HEX).
Działanie funkcji:

1. Inicjalizacja zmiennej ok:

- Zmienna ok jest ustawiana na wartość 2. Każde poprawne sprawdzenie zmniejsza tę wartość o 1.

2. Sprawdzanie dwóch ostatnich znaków w ciągu:
   - Funkcja weryfikuje dwa znaki: liczba[2] oraz liczba[3].
   - Każdy z tych znaków jest sprawdzany pod kątem zgodności z dopuszczalnymi znakami w systemie szesnastkowym:
     - Litery od A do F (lub a do f).
     - Cyfry od 0 do 9.
   - Jeśli znak spełnia warunek, zmienna ok zmniejsza się o 1.
3. Zwracanie wyniku:
   - Wartość ok po zakończeniu sprawdzania:
     - 0: obie pozycje są poprawne (dopuszczalne znaki HEX).
     - 1: jedna z pozycji zawiera niedozwolony znak.
     - 2: obie pozycje zawierają niedozwolone znaki.

Parametry:

- uint8_t * liczba: wskaźnik na ciąg znaków do sprawdzenia (wymagane, by zawierał co najmniej 4 znaki).

Zwracane wartości:

- 0: oba znaki są poprawne w systemie szesnastkowym.
- 1: tylko jeden znak jest poprawny.
- 2: oba znaki są niepoprawne.

DZIAŁANIE NA STRINGACH

```
void wybierz(uint8_t* str,uint8_t* sub,  uint8_t len) {
      strncpy(sub, str, len);
      sub[len] = 0;
}
```

void wybierz(uint8_t* str, uint8_t* sub, uint8_t len) - funkcja pobiera określoną liczbę znaków (len) z początku ciągu str i umieszcza je w ciągu sub. Na końcu utworzonego ciągu sub dodawany jest znak końca ciągu (\0), aby poprawnie zakończyć łańcuch znaków.

Działanie funkcji:

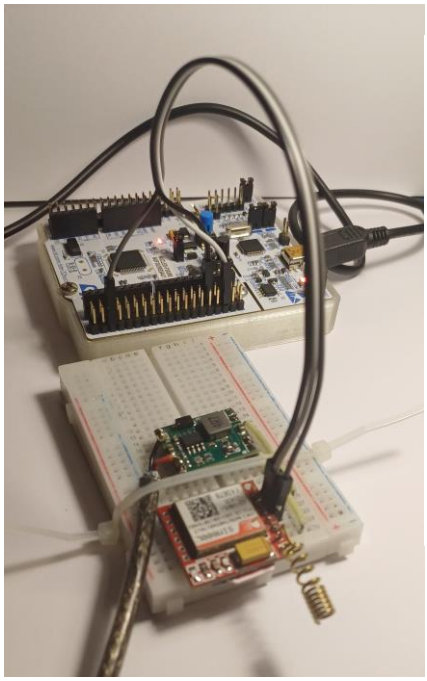
1. Kopiowanie znaków:
   - Funkcja kopiuje len pierwszych znaków z ciągu str do ciągu sub.
   - Do kopiowania używana jest standardowa funkcja strncpy().
2. Dodanie końca ciągu:
   - Na końcu nowego ciągu sub dodawany jest znak \0, który informuje, że ciąg znaków się kończy.

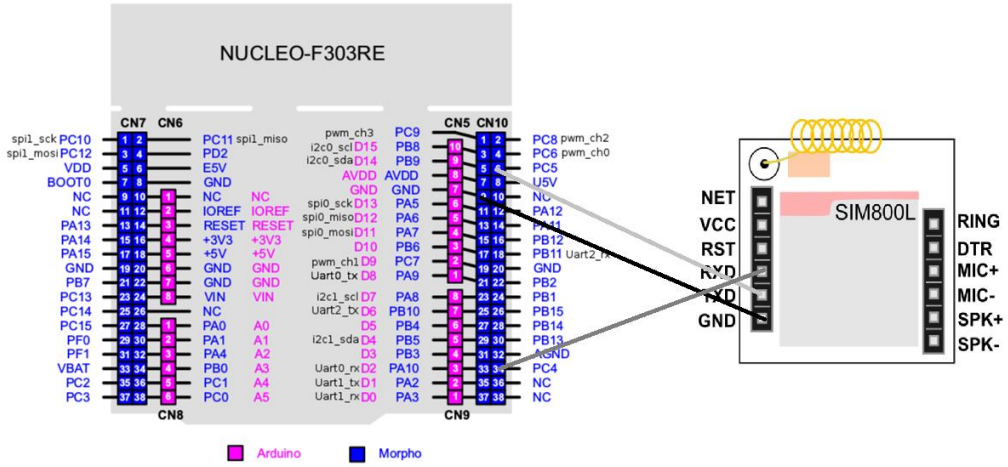
Parametry:

- uint8_t* str: wskaźnik na oryginalny ciąg znaków, z którego pobierane są dane.
- uint8_t* sub: wskaźnik na ciąg, do którego kopiowane są wybrane znaki.
- uint8_t len: liczba znaków do skopiowania z ciągu str.

HARDWARE

SCHEMAT POŁĄCZEŃ PŁYTKI STM NUCLEO-F303RE Z MODUŁEM SIM800L





Połączenie jest zrealizowane w następujący sposób:

- Masa płytki Nucleo połączona jest z masą SIM800L
- Wyjście TX USART1 połączone jest z wejściem RX modułu SIM800L
- Wyjście RX USART1 połączone jest z wyjściem TX modułu SIM800L

Moduł SIM800L ma oddzielne źródło zasilania o wartości 4V i o wydajności prądowej minimum 2 A.