POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI

PRACA DYPLOMOWA INŻYNIERSKA

RADIOWY KOMUNIKATOR TEKSTOWY

Wojciech Krzysztof Olszewski

Promotor:

Dr inż. Krzysztof Arnold

Poznań, 2018

Streszczenie

Praca pt. “Radiowy Komunikator Tekstowy” przedstawia autorski projekt bezprzewodowego urządzenia do przesyłania krótkich wiadomości tekstowych. Projekt składa się z systemu modułów, w tym Arduino Nano jako platformy oraz modułu bezprzewodowego HC-05 działającym na protokole transmisji Bluetooth.

Spis treści:

Wstęp

Komunikacja to obszerna dziedzina, obejmująca wypowiadanie myśli, transport, czy przesył cyfrowych danych. Interesującym dla nas podobszarem jest telekomunikacja, która zawiera w sobie radiokomunikację. Od czasów odkrycia elektryczności trwały prace nad sposobami transmisji informacji przez medium jakim był kabel lub szerzej linia transmisyjna. Początkowo komunikacja polegała na kluczowaniu sygnałów dźwiękowych – powstał Alfabet Morse'a. Dzięki badaniom udało się zbudować mikrofon, który pozwolił na transmisje niezmodulowanej mowy. Tak rozpoczęła się era telekomunikacji. Podjęto prace nad przesyłem jedną parą przewodów kilku sygnałów. Doprowadziło to do rozwoju modulacji cyfrowych.

Pomysł stworzenia urządzenia do tekstowej transmisji danych drogą radiową powstał z zamiłowania do radiokomunikacji. Członkostwo w krótkofalarskim kole naukowym SP3PET rozwinęło moją wiedzę i rozwiało wątpliwości co do pewnych zagadnień. Transmisja analogowa traci na znaczeniu, dlatego że jest nieefektywna w rozwijającym się świecie. W XXI wieku każdy system radiokomunikacyjny pracuje na bazie systemów cyfrowych. Logicznym następstwem jest rozwój intelektualny w tymże kierunku. Podczas tworzenia hobbystycznych projektów nabrałem wprawy w programowaniu w środowisku Arduino oraz miałem styczność z komunikacją radiową na laboratoriach, w domu (Wi-Fi, Bluetooth, DVB-T). Dlatego postanowiłem stworzyć kompletny projekt na platformie Arduino wykorzystujący moduł radiowy do transmisji danych w formie tekstowej.

Tematem pracy jest radiowy komunikator tekstowy pracujący w oparciu o moduł HC-05 i protokół transmisji Bluetooth. Sercem jest platforma Arduino i do niej podłączone są wszelkie peryferia, tj. klawiatura, wyświetlacz do wyświetlania napisanego tekstu i wcześniej wspomniany moduł HC-05. Komunikuje się on z Arduino za pomocą UART i linii RX/TX. Aby urządzenie mogło pracować potrzebne jest urządzenie lustrzane. Oba będą pracowały w trybie duplexowym, co oznacza transmisję dwukierunkową. Wybrałem protokół Bluetooth z racji rozpowszechnienia, zabezpieczeń oraz prostoty.

Celem pracy jest zbudowanie urządzenia bezprzewodowego z zasilaniem bateryjnym, umożliwiającego przesłanie wiadomości do drugiego lustrzanego urządzenia oddalonego o kilka/kilkanaście metrów od nas z możliwością podglądu wpisywanego tekstu, wpisywania dużych i małych liter. Daje to użytkownikowi dowolność w pracy na urządzeniu tak jakby pisał krótką wiadomość tekstową zwaną SMS.

W pierwszym rozdziale zostanie przedstawiona historia, tematyka, geneza oraz cel pracy. Kolejne dwa rozdziały kolejno szczegółowo mówią o platformie Arduino oraz o standardzie Bluetooth. Moduły te są kluczowe dla pracy urządzenia, dlatego poświęcono im osobne rozdziały.

Arduino wraz z mikrokontrolerem Atmega328p stanowią centrum obliczeniowe. Rozdział o platformie Arduino jest obszerny, ponieważ należy opisać funkcje, konstrukcję oraz charakterystykę modułu. Transmisja danych przebiega w standardzie Bluetooth, dlatego należy opisać standard oraz właściwości fal elektromagnetycznych wraz z antenami do tego celu wykorzystywanymi.

Treścią kolejnego rozdziału jest schemat systemu oraz opis zbudowanego urządzenia. Zawarta w nim jest także podstawa działania wykorzystanego modułu radiowego.

Piąty z kolei rozdział opisuje sposób wykorzystanie komponentów wraz z ich oprogramowaniem. Dużą częścią rozdziału są kody programu, które wraz z ich opisem przedstawiają sposób działania urządzenia pracującego w oparciu o moduł Arduino. Rozdział rozpoczyna się, od założeń projektowych, które stanowią kluczowe funkcje zaprogramowane w urządzeniu w celu jego poprawnego i bezbłędnego działania.

Przedostatni rozdział zawiera testy środowiskowe oraz programowe. W pierwszej kolejności omówione zostaje środowisko uruchomieniowe. Pierwsze testy to testy komponentów, które są wymagane w celu zweryfikowania ich poprawności działania. Kolejne testy polegają na sprawdzeniu poprawności działania urządzenia docelowego w momencie utraty zasięgu oraz w zasięgu.

Ostatni rozdział jest to podsumowanie pracy. Opisano problemy związane z projektowaniem urządzenia, zalety opracowanego komunikatora oraz możliwości dalszego rozwoju projektu.

1. Platforma Arduino

1.1. Charakterystyka platformy Arduino

Do projektu wybrano platformę Arduino ze względu na posiadane doświadczenie. Platforma początkowo skierowana była do studentów i nauczycieli, aby mogli szybko tworzyć prototypy do projektów. Rozwinęła się jednak do takiego poziomu, że stała się łatwa w użytkowaniu dla dzieci, hobbystów-amatorów, artystów, programistów, a także, mimo prostoty, dla profesjonalistów. Platforma jest licencjonowana na podstawie open-source, co znaczy, że każdy może zbudować płytkę i w nią ingerować. Fakt ten wykorzystują firmy tworząc odpowiedniki modułów Arduino, które są tańsze, pomimo konstrukcji porównywalnej z oryginałem.

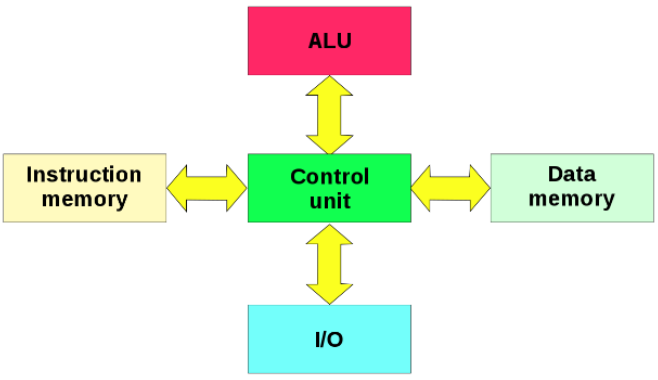
Platforma bazuje na języku zwanym *Wiring*, a środowisko oprogramowania wykorzystuje język *Processing* i nazywa się *Arduino IDE*. Arduino IDE jest wieloplatformowe, czyli współpracuje z Windows, OSX, Linux oraz od niedawna można z niego korzystać w chmurze.

Dzięki popularności baza bibliotek oraz wiedza dotycząca programowania w środowisku Arduino IDE znacząco wzrosła i może być pomocą dla tak początkujących, jak i zaawansowanych elektroników. Tysiące projektów ułatwiły programowanie, gdyż części kodów lub korzystanie z bibliotek upraszcza programowanie i zrozumienie tego, co się w kodzie dzieje.

Platforma podlega ciągłemu rozwojowi, co zachęca do jej wyboru. Na rynku istnieje wiele platform dla mikroprocesorów, lecz ta rozwija się najszybciej i jest przy tym najtańsza, co zawdzięcza licencji open-source. W jej skład wchodzi też szeroki wachlarz „nakładek" zwanych shieldami, które zwiększają funkcjonalność albo posiadają gotowe moduły, które wystarczy oprogramować według własnych potrzeb lub skorzystać z gotowych kodów [1].

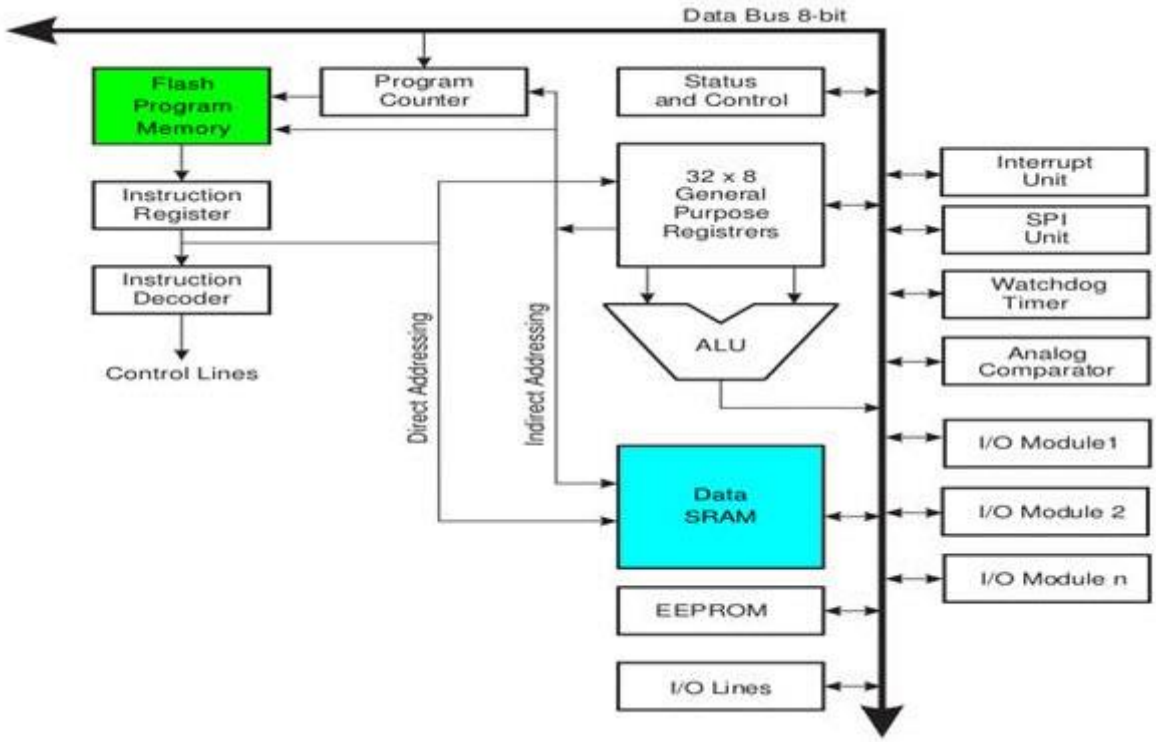
1.2. Architektura modułu Arduino Nano

Mikrokontroler na platformie Arduino wykorzystuje tzw. *Architekturę Harvardzką*, co znaczy, że pamięć podzielona jest na dwie osobne pamięci dla danych i programu (rys. 1.1).



Rys. 1.1 Schemat architektury harvardzkiej [2]

Program wykorzystuje pamięć Flash, a dane pamięci SRAM i EPROM. Rozszerzeniem rysunku 1.1 przedstawiającym kompletną architekturę rdzenia AVR jest rysunek 1.2.



Rys. 1.2 Architektura AVR [3]

Mikrokontroler w nowych wersjach platformy to Atmega328 charakteryzujący się 32kb pamięci flash oraz 2kb SRAM i 1kb EPROM. Zegar działa na częstotliwości 16MHz [4].

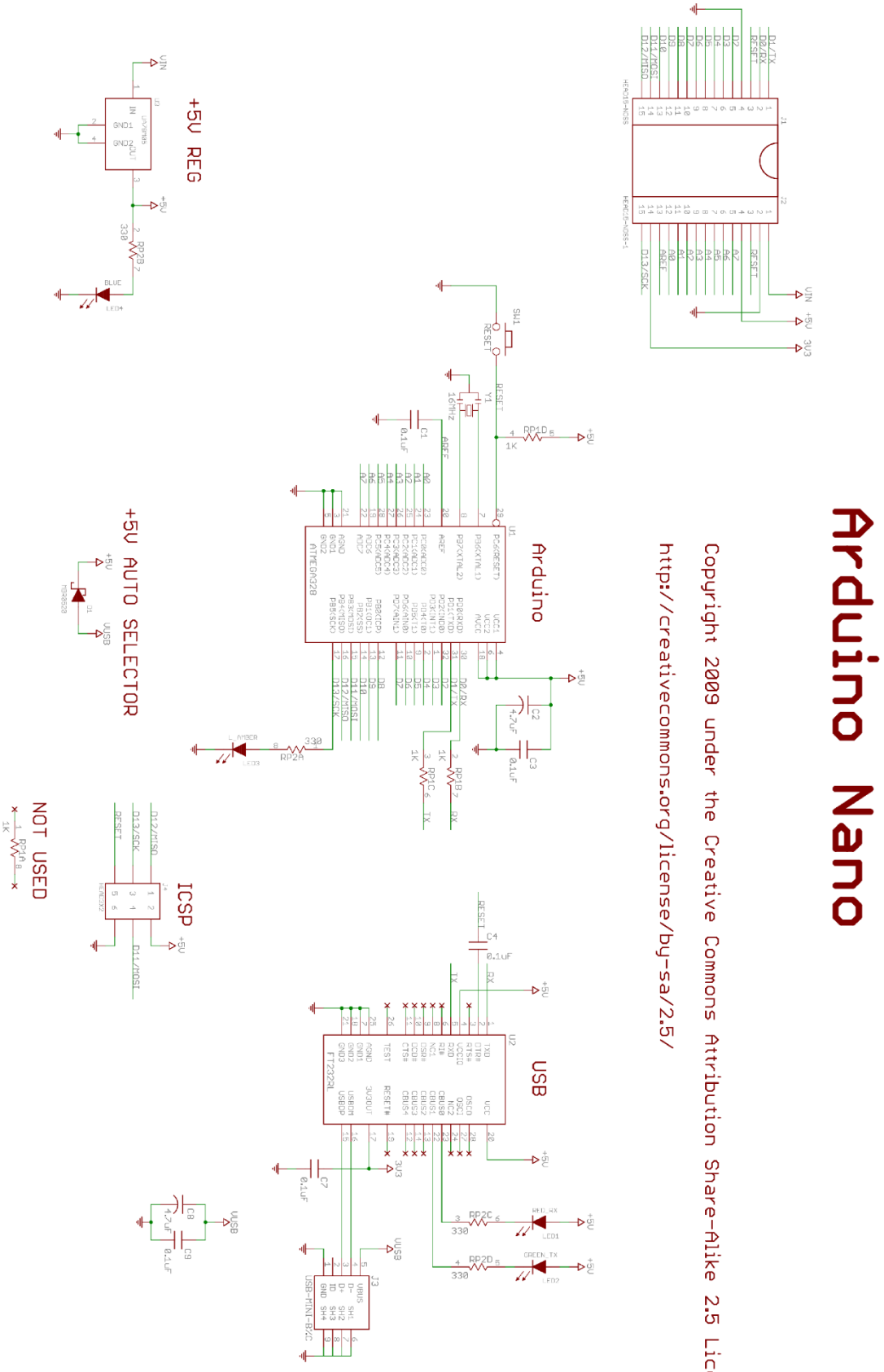
Konstrukcję wraz z podzespołami platformy Arduino Nano przedstawia schemat ideowy platformy przedstawiony na rysunku 1.3.

1.3. Właściwości mikrokontrolera Atmega328p

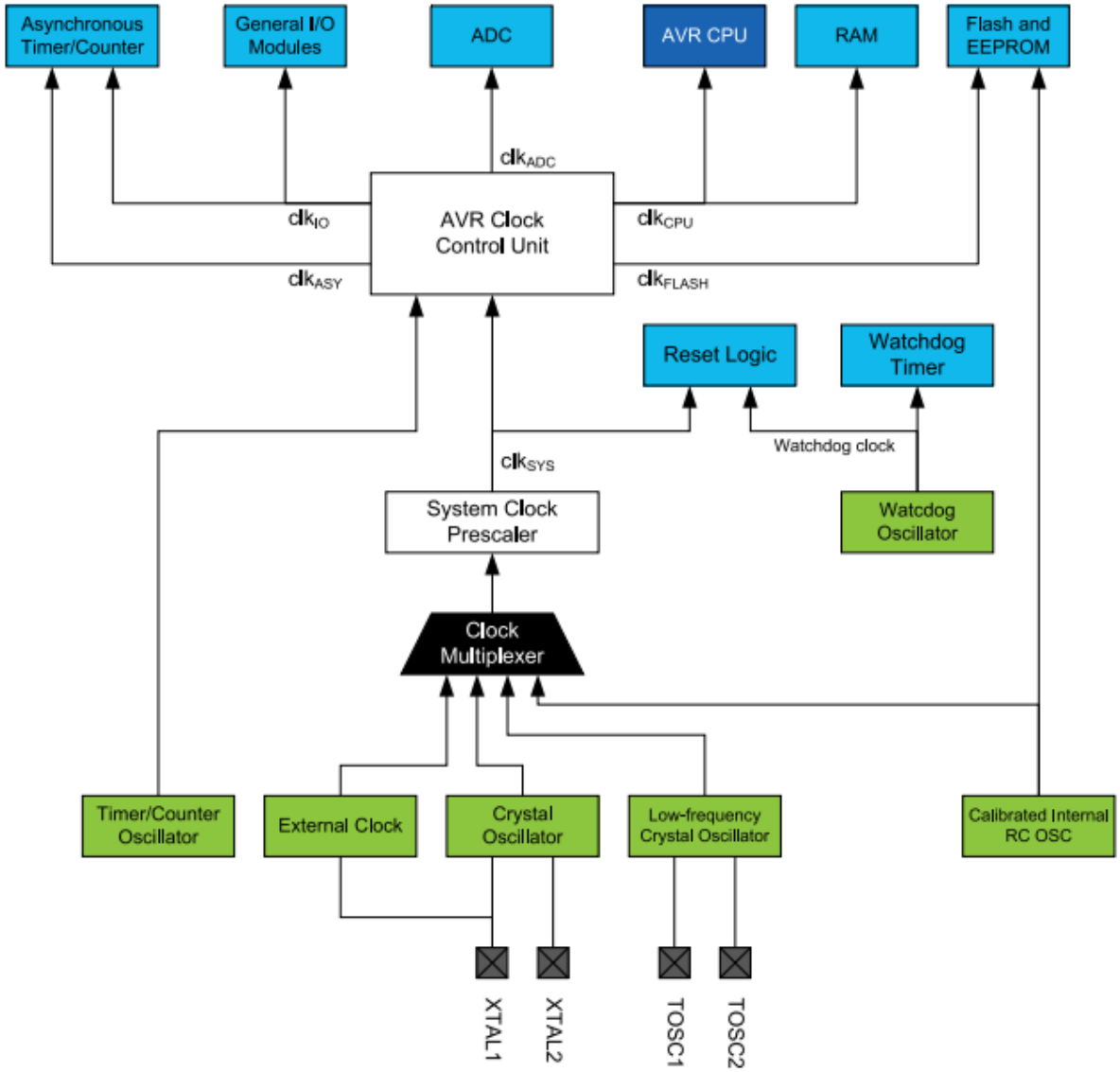
1.3.1. Zegar

Rysunek 1.4 przedstawia zasadę działania systemu zegarowego w urządzeniu i jego rozprowadzanie. Żaden z zegarów nie musi być aktywny w danym czasie. Dlatego aby zredukować zużycie energii, zegary do modułów, które nie są w użyciu mogą być nieprzesyłane dzięki trybom uśpienia.

Częstotliwość bazowa systemu jest to częstotliwość generowana przez tzw. System Clock Prescaler. Należy tutaj zwrócić uwagę, że wszystkie wyjścia z AVR Clock Control Unit posiadają tę samą częstotliwość [5]



Rys 1.3 Schemat Arduino Nano [5]



Rys. 1.4 Rozprowadzanie zegara [6]

1.3.2. Przerwania

W Atmega328p mamy 2 rodzaje przerwań: wewnętrzne i zewnętrzne. Zewnętrzne przerwania mogą być skonfigurowane tak, aby wyzwalać jednym z 4 stanów. Poziom niski zgłasza przerwanie, wtedy, kiedy moduł wykryje na pinie napięcie na poziomie bliskim masie. Każda zmiana stanu logicznego załącza przerwanie, tj. zmiana ze stanu wysokiego (Vcc) na niski (masa) albo odwrotnie. Jeśli nastąpi zmiana napięcia na pinie, to na zboczu opadającym przerwanie zmieni swój stan z wysokiego na niski i odwrotnie dla zbocza narastającego. Ustawienia tych pinów i kombinacje przedstawia tabela 1.1. Jest możliwość skonfigurowania każdego pinu INTx niezależnie [6].

Zewnętrzne przerwania wykorzystują 3 rejestry (tabela 1.2, tabela 1.3, tabela 1.4) [6].

Tabela 1.1 Ustawienia ISC Bit [7]

ISCx1	ISCx0	OPIS
0	0	Niski poziom INTx generuje przerwanie
0	1	Dowolna zmiana stanu INTx generuje przerwanie
1	0	Opadające zbocze INTx generuje przerwanie
1	1	Narastające zbocze INTx generuje przerwanie

Tabela 1.2 External Interrupt Mask Register [7]

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EIMSK	-	-	-	-	-	-	INT1	INT0

Tabela 1.3 External Interrupt Flag Register [7]

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EIFR	-	-	-	-	-	-	INTF1	INTF0

Tabela 1.4 External Interrupt Control Register A [7]

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
EICRA	-	-	-	-	ISC11	ISC10	ISC01	ISC00

1.3.3. Wejścia/wyjścia

Każdy port mikroprocesora używany, jako cyfrowy posiada funkcjonalność odczytu/modyfikacji/zapisu. Każde wyjście jest na tyle trwałe i silne, żeby mogło zasilić wyświetlacze LED oraz diody LED bezpośrednio. Porty mają indywidualnie rezystory podciągające, służące do podciągania napięcia oraz diody chroniące przed przeciążeniem do masy i zasilania. Pojedynczy pin posiada swoje własne rejestry: DDxn, PORTxn, PINxn (rys. 3.4).

Załączanie pinu polega na wystawieniu logicznej jedynki w rejestrze PORTxn przy wartości 1 w rejestrze DDRxn.. Aby to osiągnąć możemy użyć pojedynczej instrukcji SBI, która pozwoli nam wystawić pojedynczą jedynkę na porcie.

Przełączanie pomiędzy wejściem/wyjściem polega na wystawianiu na wejścia rejestrów odpowiednich kombinacji bitów. Kombinacje te przestawia tabela 1.5.

Tabela 1.5 Konfiguracja portów pinów [6]

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

1.3.4. Liczniki

Mikroprocesor posiada trzy timery/liczniki ogólnego zastosowania.

1) 8 bitowy TC0 cechuje się [6]:
   - dwoma niezależnymi wyjściami używanymi w trybie CTC,
   - podwójnie buforowanym wyjściem porównującym rejestry,
   - timerem czyszczącym po poprawnym porównaniu,
   - generatorem PWM,
   - zmiennym okresem PWM,
   - generatorem częstotliwości,
   - trzema niezależnymi źródłami przerwań.
2) 16 bitowy TC1 cechuje się [6]:
   - dwoma niezależnymi wyjściami używanymi w trybie CTC,
   - podwójnie buforowanym wyjściem porównującym rejestry,
   - timerem czyszczącym po poprawnym porównaniu,
   - generatorem PWM,
   - zmiennym okresem PWM,
   - generatorem częstotliwości,
   - trzema niezależnymi źródłami przerwań,
   - jedną jednostką przechwytującą wejście,
   - niwelatorem szumu wejściowego,
   - zewnętrznym licznikiem zdarzeń.
3) 8 bitowy TC2 cechuje się [6]:
   - licznikiem kanałowym,
   - timerem czyszczącym po poprawnym porównaniu,
   - generatorem PWM,
   - generatorem częstotliwości,
   - 10 bitowym preskalerem zegara,
   - trzema źródłami przerwań,
   - zezwoleniem na użycie 32 kHz zewnętrznego sygnału zegarowego.

Liczniki TC0 i TC1 używają tego samego preskalera, ale mają różne jego ustawienia [10].

1.3.5. Komunikacja USART

USART – The Universal Synchronous and Asynchronous Serial Receiver and Transmitter jest wysoce elastycznym narzędziem do komunikacji szeregowej przewodowej. Może być także być używany w trybie Master SPI [6].

Cechuje się on [6]:

- pełną pracą duplexową – niezależnym odbiorem i nadawaniem,
- trybem pracy synchronicznym i asynchronicznym,
- trybem pracy Master albo Slave,
- generatorem szybkości transmisji wysokiej rozdzielczości,
- wsparciem ramek szeregowych 5, 6, 7, 8, 9 bitowych z 1 lub 2 bitami stopu,
- generatorem bitów parzystości i ze sprzętowym sprawdzaniem parzystości,
- sygnalizacja błędu Data Overflow,
- detekcją błędu ramek,
- filtracją szumu,
- trzema oddzielnymi przerwaniami TX Complete, TX Data Register oraz RX Complete,
- komunikacją między procesorową,
- asynchronicznym trybem komunikacji o podwójnej szybkości.

1.4. Funkcje modułu Arduino Nano

Funkcje modułu Arduino Nano są tożsame z funkcjami Atmega328p.

Podstawowe piny to:

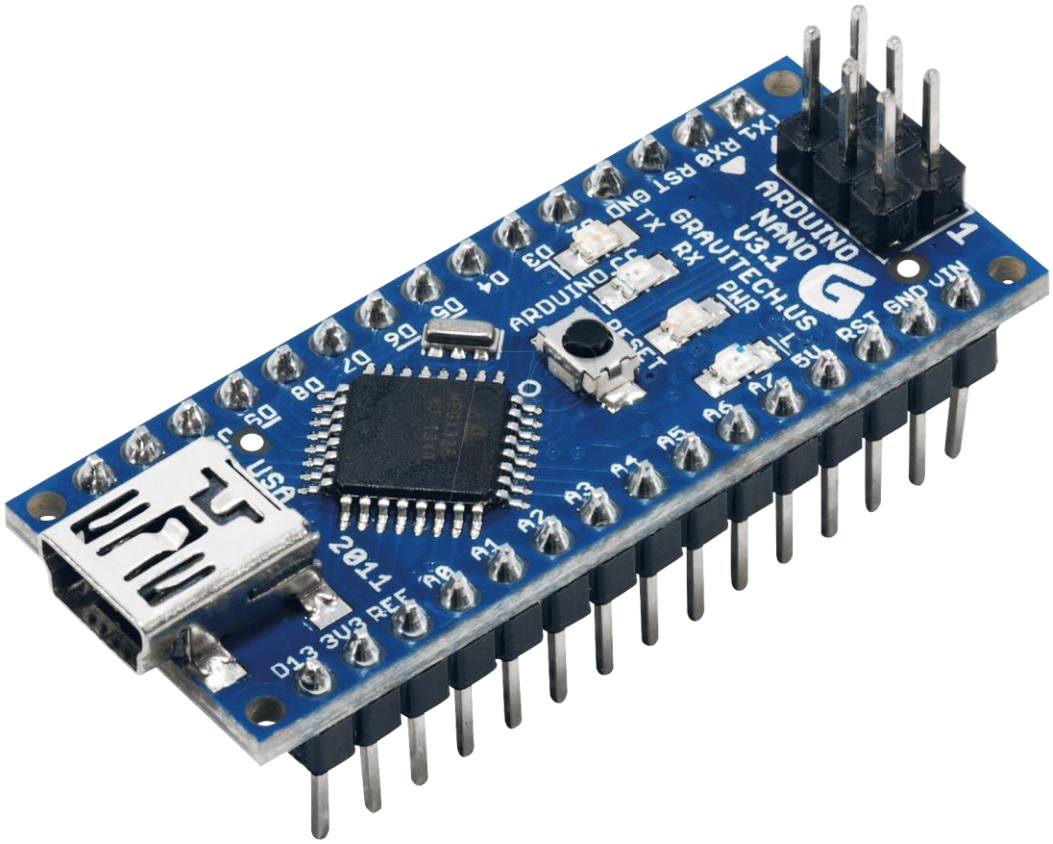
- cyfrowe piny odczytujące logiczne wartości napięcia z pinu,
- cyfrowe piny nadające logiczne wartości napięcia na pin,
- analogowe piny odczytujące wartości napięcia z pinu za pomocą przetwornika ADC,
- analogowe piny nadające analogowe wartości napięcia na pin za pomocą generatora PWM,
- wyjście 3,3V,
- UART.

Arduino Nano dodatkowo pozwala na:

- programowanie modułu za pomocą kabla USB, posiada konwerter USB-UART,
- programowanie w języku C w środowisku Arduino IDE,
- sygnalizację stanu za pomocą LED,
- resetowanie mikroprocesora,
- zabezpieczenie przed przeciążeniem (specjalizowana dioda).

1.5. Konstrukcja modułu

Arduino Nano korzysta z mikroprocesora Atmega328 oraz zbudowany jest na dwuwarstwowej płytce PCB wraz z dodatkowymi goldpinami do montażu powierzchniowego (rys. 1.5).

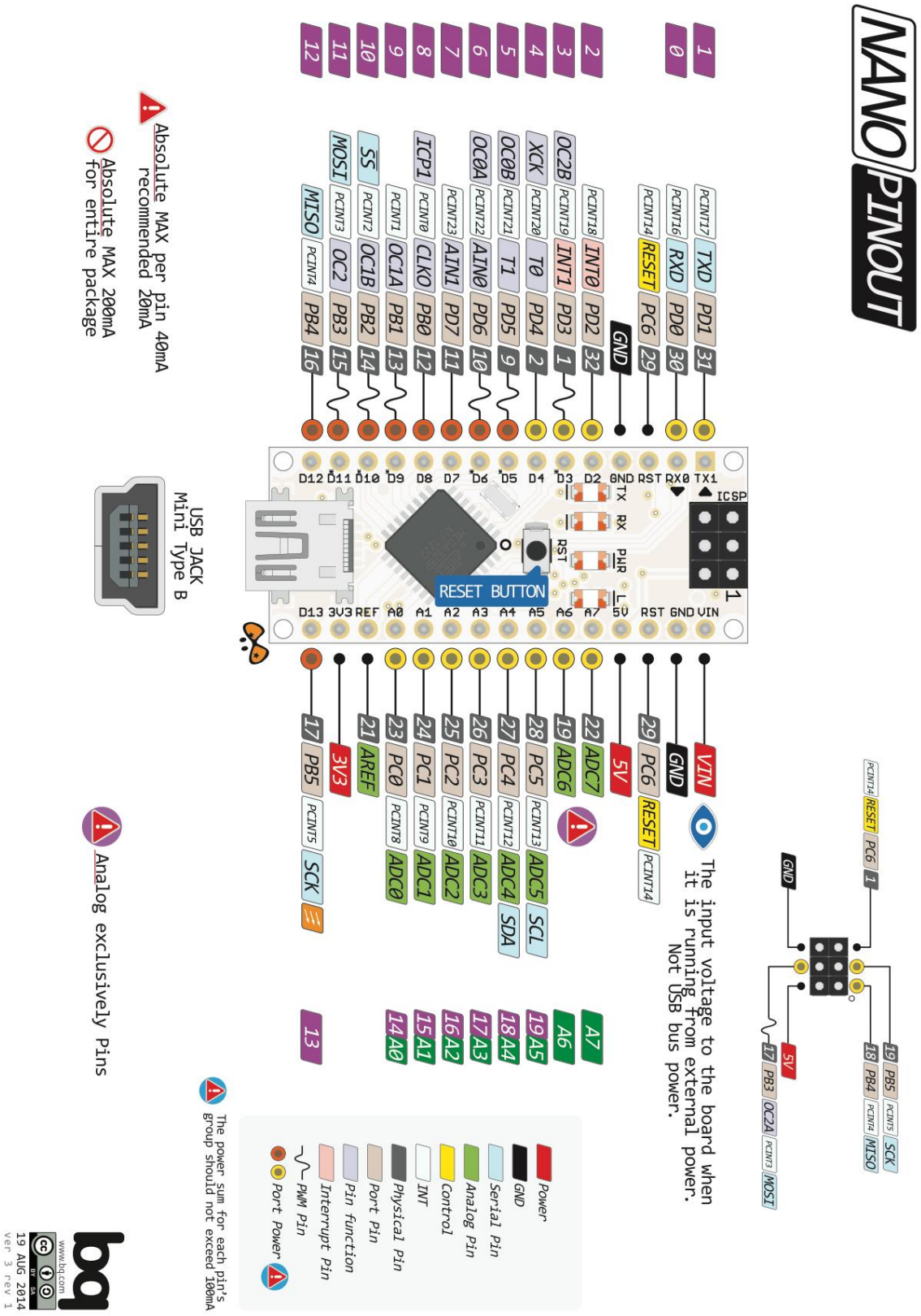


Rys 1.5 Moduł Arduino Nano [8]

Urządzenie posiada:

- 12 portów cyfrowych wejścia/wyjścia z rezystorami podciągającymi,
- na portach cyfrowych 6 wyjść generatora PWM,
- 8 portów analogowych, w tym 6 wejścia/wyjścia z rezystorami podciągającymi i 2 wejścia bez rezystorów podciągających,
- 24 piny przerwań typu PCINT (zmiana końcówki),
- 22 piny podzielone na porty 8-bitowe,
- piny do zasilania modułu ze źródła zewnętrznego: VIN i GND,
- piny zasilające GND, 5V, 3V3,
- diody informacyjne oraz 1 diodę połączoną z pinem D13,
- stabilizator napięcia,
- diodę zabezpieczającą zasilanie,
- programator CH340,

Schemat modułu pokazany jest na rysunku 1.6.



Rys 1.6 Schemat pinów Arduino Nano [9]

2. Standard Bluetooth

2.1. Częstotliwości i prawo telekomunikacyjne

Sygnały radiowe przekazywane są za pośrednictwem nośników, jakimi są fale elektromagnetyczne. Fale te podlegają różnym zjawiskom fizycznym, takim jak refrakcja, dyfrakcja, tłumienie itd. Podzielono je na podgrupy w zależności od długości fali (częstotliwości) ze względu na sposoby ich rozchodzenia się [10]:

- **fale długie** - zakres od 10 do 100 kHz, fale te mają olbrzymie zasięgi dochodzące do kilku tysięcy kilometrów, co wynika z tego, że Ziemia i niższe warstwy jonosfery działają dla nich jak falowód,

- **fale średnie** – zakres od 100 kHz do 1500 kHz, fale mają zasięg rzędu setki kilometrów w dzień, w nocy mają trochę większy z racji odbić od jonosfery, ulegają też wielu szkodzącym im zjawiskom fizycznym (zaniki interferencyjne, selektywne itd.),

- **fale pośrednie** – zakres od 1,5 MHz do 3 MHz, fale o charakterze przejściowym pomiędzy falami średnimi a krótkimi, wyższe częstotliwości nie są jednak w dzień tłumione przez jonosferę,

- **fale krótkie** – zakres od 3 MHz do 30 MHz, fale wykorzystywane, jako powierzchniowe oraz jonosferyczne (światowy zasięg), z tym, że wpływ mają tutaj zaburzenia jonosfery, czyli pogoda słoneczna i nuklearna działalność człowieka,

- **fale ultrakrótkie –** zakres od 30 MHz do 300 MHz, propagują prostolinijnie, jonosfera ich nie odbija, zachodzi retrakcja (ugięcie fali w atmosferze), która pozwala mieć większy zasięg niż zasięg widzenia [11],

- **mikrofale** – zakres od 300 MHz (w praktyce od ok. 1 GHz) do 300 GHz, fale rozchodzą się zasadniczo prostoliniowo, tłumione są przez parę wodną, są odbijane od obiektów (jest to zależne od wielkości obiektu) [12].

Mikrofale mają bardzie duże zastosowanie z racji szerokości pasma. Dla częstotliwość 10 GHz 1% pasma to 100 MHz. Pozwalają one na zwiększenie liczby kanałów w danym paśmie. Jest to bardzo atrakcyjne w dobie cyfryzacji. Niestety konwencjonalne metody projektowania układów nie są tutaj przydatne, gdyż obwody są porównywalne z długościami fal [12]

W Polsce zasady korzystania z częstotliwości reguluje Urząd Komunikacji Elektronicznej. Moduł Bluetooth korzysta z mikrofalowego przedziału częstotliwości zwanego ISM (*Industrial, Scientific, Medical*). Według UKE przedział ten mieści się pomiędzy 2400 MHz a 2483,5 MHz i jest przeznaczony „do celów przemysłowych, naukowych, medycznych, domowych oraz im podobnych z wyłączeniem zastosowań do celów telekomunikacyjnych” [13]. Oznacza to, że użytkownik domowy może wykorzystywać to pasmo do własnych potrzeb. W przedziale mamy 79 kanałów po 1 MHz.

Podział pasm mikrofalowych obrazują tabele poniżej [14].

Stare oznaczenia

ELF	30 – 300 Hz
VF	300 – 3000 Hz
VLF	3 – 30 kHz
LF	30 – 300 kHz
MF	0,3 – 3 MHz
HF	3 – 30 MHz
VHF	30 – 300 MHz
UHF	300 – 1000 MHz
L	1 – 2 GHz
S	2 – 4 GHz
C	4 – 8 GHz
X	8 – 12,4 GHz
Ku	12,4 – 18 GHz
K	18 – 26 GHz
Ka	26 – 40 GHz
Pasmo milimetrowe	40 – 100 GHz

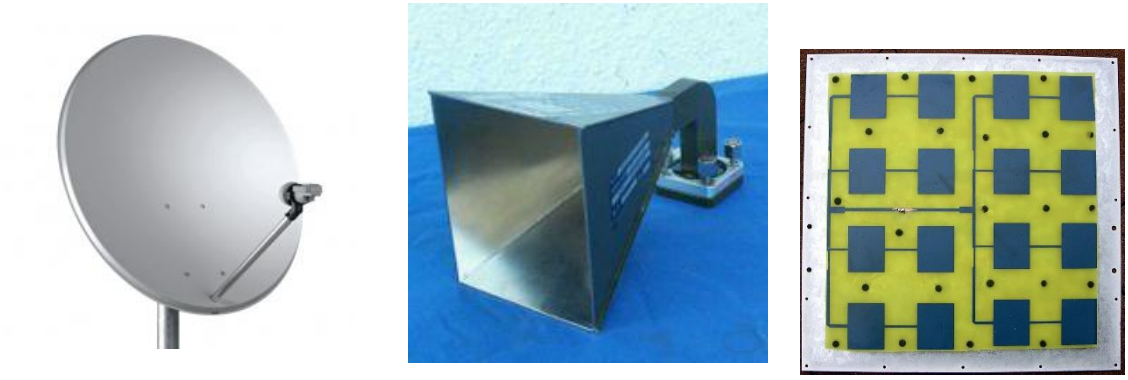
Nowe oznaczenia:

A	0 – 250 MHz
B	250 – 500 MHz
C	500 – 1000 MHz
D	1 – 2 GHz
E	2 – 3 GHz
F	3 – 4 GHz
G	4 – 6 GHz
H	6 – 8 GHz
I	8 – 10 GHz
J	10 – 20 GHz
K	20 – 40 GHz
L	40 – 60 GHz
M	60 – 100 GHz

2.2. Anteny mikrofalowe

Do najczęściej stosowanych anten mikrofalowych zaliczamy [15]:

- Anteny paraboliczne (rys. 2.1a). – charakteryzują się dużą kierunkowością (szpilkową) i olbrzymim zyskiem energetycznym rzędu 50 dB. Wykorzystywane są ze względu na wymagane małe moce nadajników względem odległości. Promień anteny jest tak dobrany, aby skupiał wszystkie promienie padające w jednym miejscu, w ognisku. Ustawiając w tym miejscu źródło oświetlające przechwytujemy prawie całą energię
- Anteny tubowe (rys. 2.1b). – zbudowane są z otwartego na końcu falowodu o przekroju kołowym lub prostokątnym z modyfikacją przy otworze, aby wyeliminować silne odbicie. Posiadają duży zysk energetyczny rzędu kilkunastu dB, mały WFS, szerokie pasmo i prostą konstrukcję. Antena tubowa wykorzystywana jest w antenach parabolicznych jako promiennik.
- Anteny mikropaskowe (rys. 2.1c). – można je umieścić na różnych powierzchniach, dzięki małym rozmiarom. Są tanie w produkcji oraz łatwo się integrują z układami elektronicznymi. Mają małą szerokość pasma pracy, promieniują z obwodów zasilających oraz nie są odporne na zakłócenia.



a) Antena paraboliczna [16] b) Antena tubowa [17] c) Antena mikropaskowa [18]

Rys. 2.1 Anteny mikropaskowe

2.3. Opis standardu

Standard Bluetooth, tj. IEEE 802.15.1 został wprowadzony, aby umożliwić współpracę urządzeń bezprzewodowych. Wyparł on mało efektowny standard IrDA, który polegał na transmisji danych na podczerwonej fali świetlnej. Wymuszało to bezpośrednie widzenie się urządzeń i było mało wygodne.

Bluetooth pozwala na zbudowanie osobistej sieci bezprzewodowej (PAN – *Personal Area Network*) z zasięgiem dookólnym około 10 m lub 100 m w zależności od modułu. Pracuje on, jak wspomniano wcześniej na częstotliwości 2,4 GHz, to znaczy w paśmie mikrofalowym. Fale te są odbijane i mają zasięg LOS (Line-of-Sight) tj. wzrokowy. Pozwalają na transfer danych w relatywnie szerokich kanałach. Do transmisji wykorzystywane są anteny mikropaskowe ze względu na małe wymiary i akceptowalne zasięgi.

Standard przewiduje wiele prędkości w zależności od wersji zaczynając od 2,1 Mb/s dla dość powszechnej wersji 2.1 + EDR, kończąc na 50 Mb/s dla wersji 5.0. W zależności od wersji wykorzystywane są też różne metody modulacji cyfrowej. Transmisja jest duplexowa synchroniczna ze zwielokrotnieniem czasowym TDMA. Aby zmniejszyć zakłócenia wykorzystano metodę rozpraszania widma FHSS (*Frequency-Hopping Spread Spectrum*), która zmienia nośną 1600 razy na sekundę oraz zmniejszono je poprzez nadmiarowość w ramkach.

Nagłówek w pakietach EDR jest nadawany zawsze za pomocą kluczowania częstotliwości. Za nagłówkiem jest przedział ochronny 5µs, aby zabezpieczyć moduł w czasie przełączania modulacji przed błędami. Następnie wysyłana jest sekwencja synchronizacyjna o czasie trwania 11µs oraz realizowana transmisja bitów informacyjnych. Jeśli zapotrzebowanie na przepływność jest małe, pakiety EDR mogą zmniejszyć swoją długość i pozwolić przez to zaoszczędzić energię, która w urządzeniach zasilanych bateryjnie lub akumulatorowo jest ograniczona [19].

3. Projekt nadajnika i odbiornika tekstu

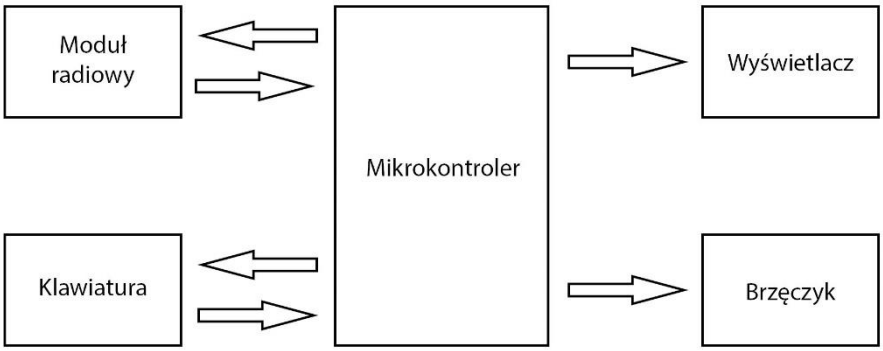
3.1. Założenia projektowe

Podstawowe wymagania stawiane projektowi:

- wykorzystanie zasilania bateryjnego, aby można było zrezygnować z kabli zasilających, jest to jeden z celów projektu,
- przekazywanie znaków alfanumerycznych z podglądem wysyłanej i odbieranej informacji,
- zasięg transmisji do kilkunastu metrów,
- komunikacja bezprzewodowa,
- opcjonalne użycie brzęczyka jako informatora wiadomości i sygnału wciśnięcia przycisku,
- zabezpieczenie przycisków na klawiaturze przed drganiami styków oraz zablokowanie przytrzymania przycisku,
- łącznie się modułów tylko i wyłączne ze sobą,
- wykonanie urządzeń na płytce PCB.

3.2. Koncepcja systemu

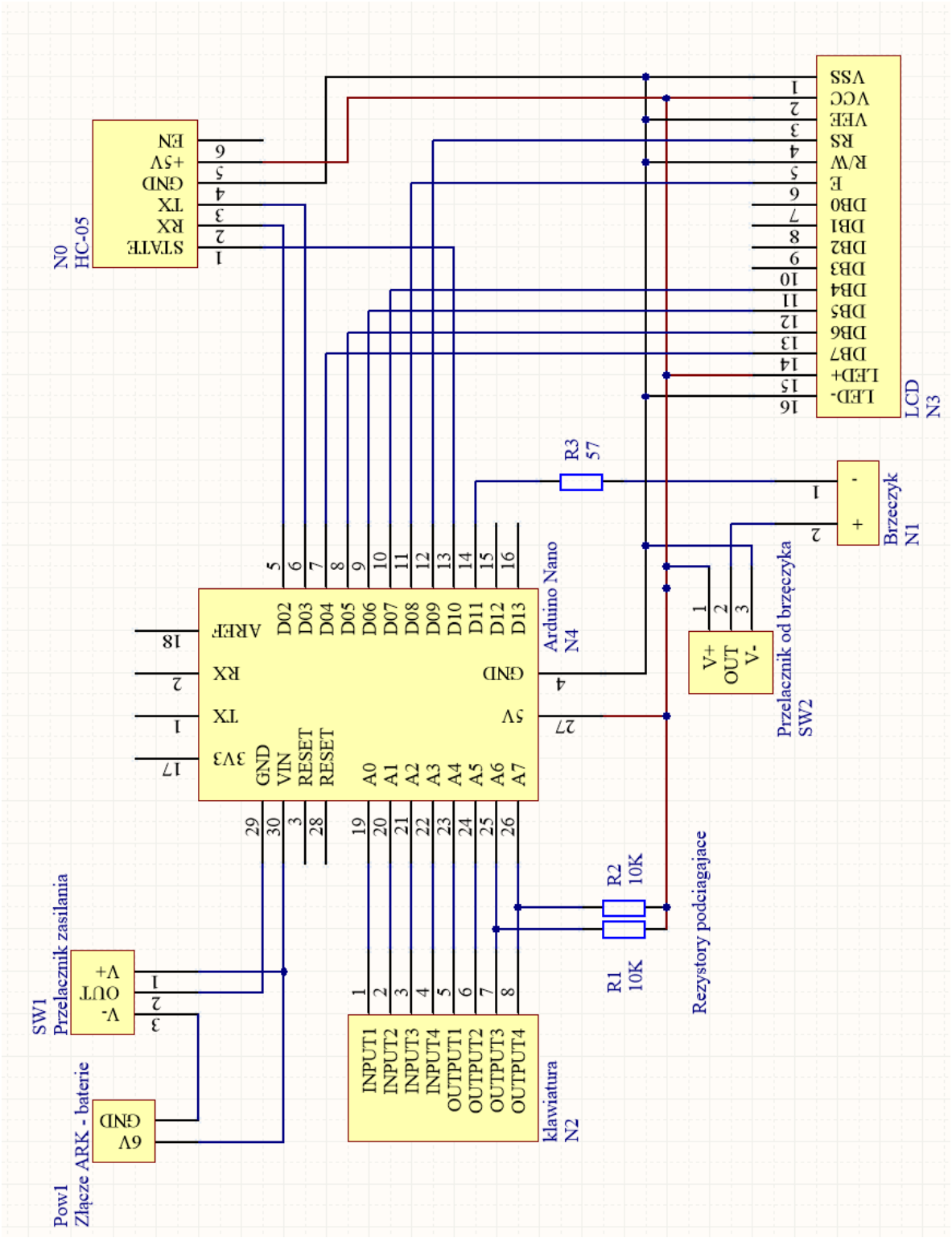
W skład systemu wchodzą: mikrokontroler, moduł radiowy, klawiatura, wyświetlacz oraz opcjonalnie brzęczyk (rys. 3.1).



Rys. 3.1 Koncepcja systemu

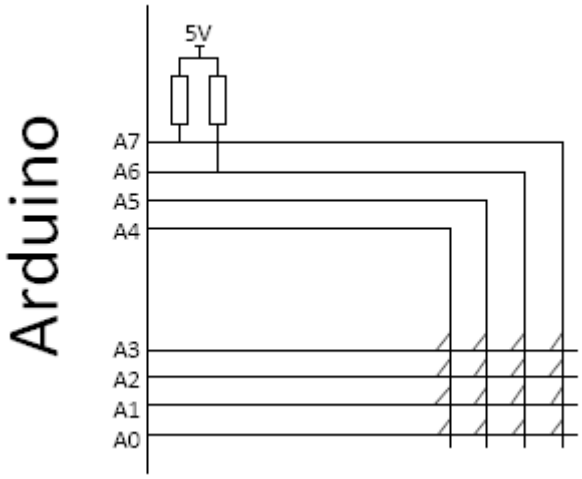
Użytkownik wpisuje tekst na klawiaturze, jednocześnie śledząc pojawianie się wprowadzanych znaków na wyświetlaczu LCD. Otrzymywane są jednocześnie sygnały z brzęczyka potwierdzające wciśnięcie przycisku.

Schemat ideowy systemu wraz z połączeniami przedstawia rysunek 3.2.



Rys. 3.2 Schemat ideowy systemu

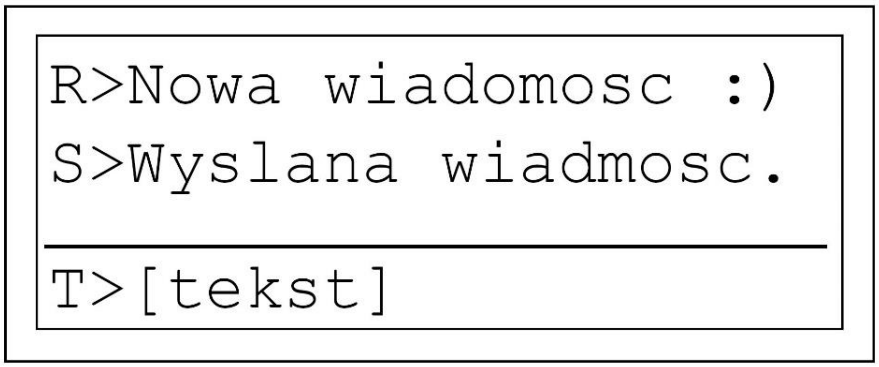
Klawiatura matrycowa 4x4 jest sterowana sekwencyjnie, linia po linii (rys. 3.3).



Rys. 3.3 Podłączenie klawiatury

Moduł radiowy wysyłając i odbierając dane musi pracować w trybie duplexowym, tak samo jak klawiatura. Mikrokontroler na pinach analogowych przeznaczonych dla klawiatury wystawia logiczną zero tylko na jedno wejście i sprawdza, na którym wyjściu się ono pojawi. Brzęczyk podłączony do zasilania z jednej strony czeka na pojawienie się logicznego zera na pinie z mikroprocesora.

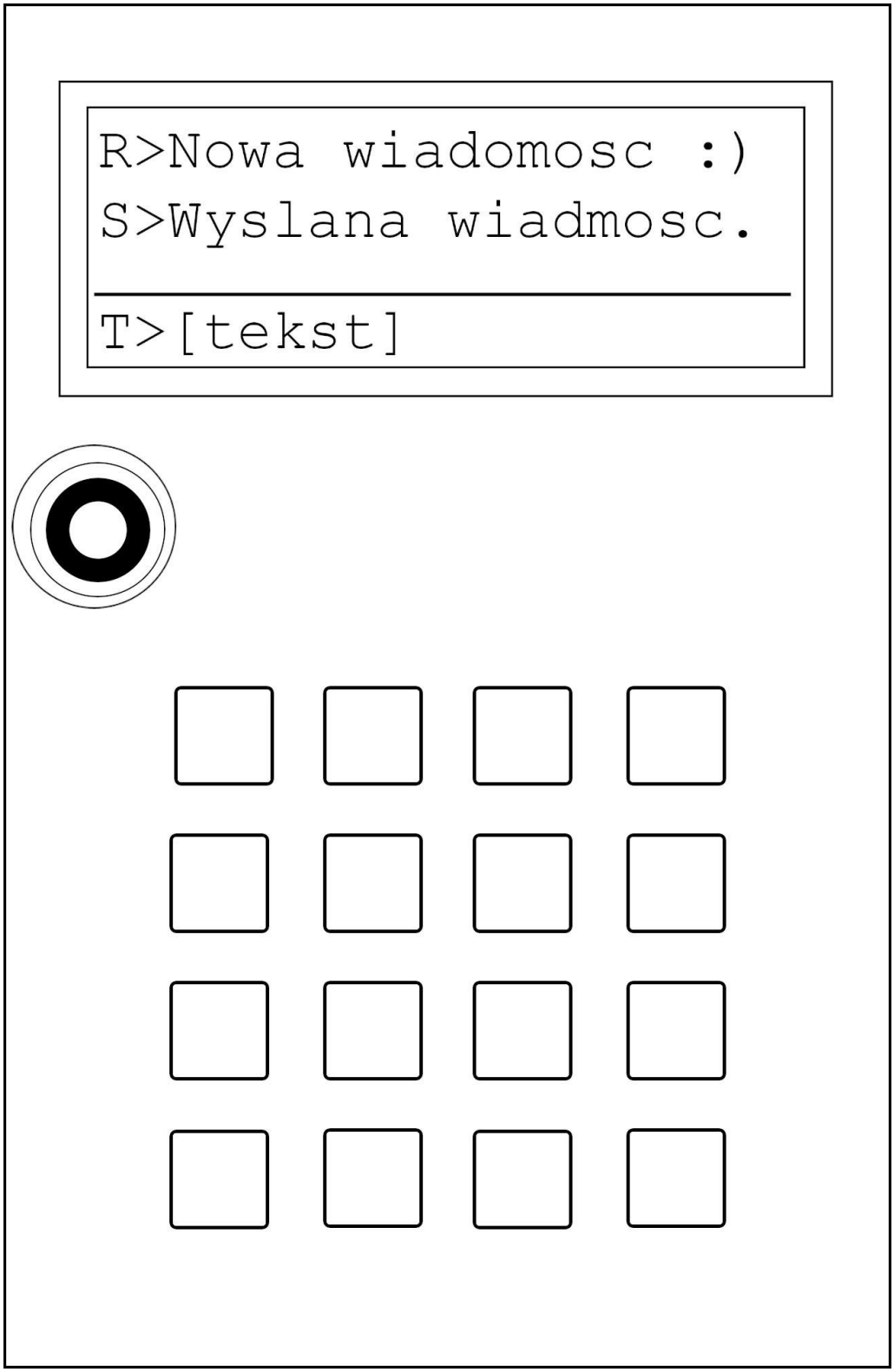
Koncepcję interfejsu użytkownika wyświetlanego przedstawia rysunek 3.4.



Rys. 3.4 Koncepcja interfejsu

Interfejs pokazany na rysunku 4.2 przedstawia dwie ostatnie wiadomości w relacji: odebrana i odebrana, odebrana i wysłana, wysłana i odebrana lub wysłana i wysłana, w zależności od tego jak kolejno wiadomości zostały przetworzone. Założeniem jest to, aby najnowsza wiadomość była zawsze na górze wyświetlacza.

Koncepcję wyglądu urządzenia ilustruje rysunek 4.5. Jest to wstępny schemat i może on ulec zmianie w trakcie tworzenia i projektowania urządzenia.



Rys. 3.5 Koncepcja urządzenia

3.3. Opis działania modułu odbiorczo-nadawczego

Protokół Bluetooth tworzy piko sieć, w której może uczestniczyć do 8 urządzeń. Wymagane jest, aby jeden moduł nadrzędny pracował w trybie Master, a pozostałe w trybie Slave. Od tego momentu jedno z urządzeń pracujące w trybie Master będzie nazywane *RTTmaster*, a drugie w trybie Slave *RTTslave*.

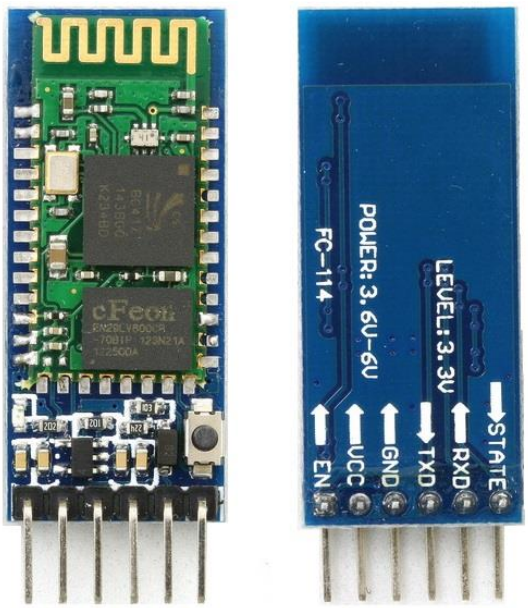
Oba wykorzystane moduły radiowe HC-05 (rysunek 3.6) należy skonfigurować do trybu parowania wg adresów z listy. Powoduje to, że osoba nieznająca hasła nie ma możliwości podłączenia się do piko sieci. Do modułu RTTslave został dodany adres urządzenia RTTmaster w celu utworzenia połączenia. Adresy obu modułów zostały odczytane pojedynczo za pomocą odpowiednich komend (tabela 3.1). Kody i odpowiednie funkcje zostaną przedstawione w rozdziale 4.3.

Tabela 3.1 Adresy modułów HC-05

RTTslave	98d3:31:fc4ad2
RTTmaster	98d3:32:30dd05

Zasada działania modułu polega na tym, że w mikroprocesorze wysyłamy dane za pomocą UART na linie TX i RX mikroprocesora. Moduł Bluetooth odbiera dane za pomocą linii TX oraz RX w relacji TX mikroprocesora podłączony do RX modułu i RX mikroprocesora do TX modułu w konwencji krzyżowej. Wiadomości wczytywane są do buforów, gdzie czekają na wysłanie przez układ nadawczy. W chwili utraty połączenia z drugim modułem wiadomości pozostają w buforach i czekają na wznowienie transmisji. W drugim module RTTslave dopóki wiadomość, dopóki niezostanie odczytana pozostaje w buforze.

Wyprowadzenie STATE wykorzystywane będzie w projekcie, jako wskaźnik połączenia. W chwili utraty połączenia pin przyjmie wartość logicznego zera. Pin EN pozwala programować moduł poprzez tryb AT.



Rys. 3.6 Moduł HC-05 (inaczej FC-114) wraz z wyprowadzeniami [20]

4. Oprogramowanie urządzenia

4.1. Założenia projektowe

Funkcje wymagające zaprogramowania i zaimplementowania to:

- wyświetlanie dwóch ostatnich wiadomości oraz najnowszej na górze.
- wiadomości 18 znakowe.
- odczyt wciśniętego przycisku.
- wybieranie literki w pętli.
- wybór dużej lub małej litery.
- usuwanie litery.
- zatwierdzanie litery po odpowiednim czasie.
- czyszczenie ekranu: ostatnich wiadomości wraz z wpisywaną wiadomością.
- wiadomości informacyjne.
- wyświetlanie przychodzącej wiadomości podczas wpisywania tekstu,
- komunikaty dźwiękowe,
- odpowiednie zabezpieczenia,
- tryb debugowania – po podłączeniu USB można w konsoli podejrzeć informacje dodatkowe,
- tryb szybkiego wpisywania znaków na klawiaturze.

Zabezpieczenia:

- przed usunięciem wybieranej litery (niezatwierdzonej),
- przed wysłaniem wiadomości w momencie wybierania litery,
- przed przytrzymaniem przycisku – trzymanie przycisku oznacza jedno wciśnięcie,
- przed wybraniem znaku powyżej limitu.

Wiadomości informacyjne:

- „connected” – w przypadku nawiązania połączenia,
- „connection lost” – w przypadku utraty połączenia spowodowanej zbyt dużym oddaleniem się lub wyłączenia jednego z urządzeń,
- „ABC/abc” – w przypadku zmiany wielkości liter,
- „limit znaków” – w przypadku dojścia do końca ekranu z kursorem.

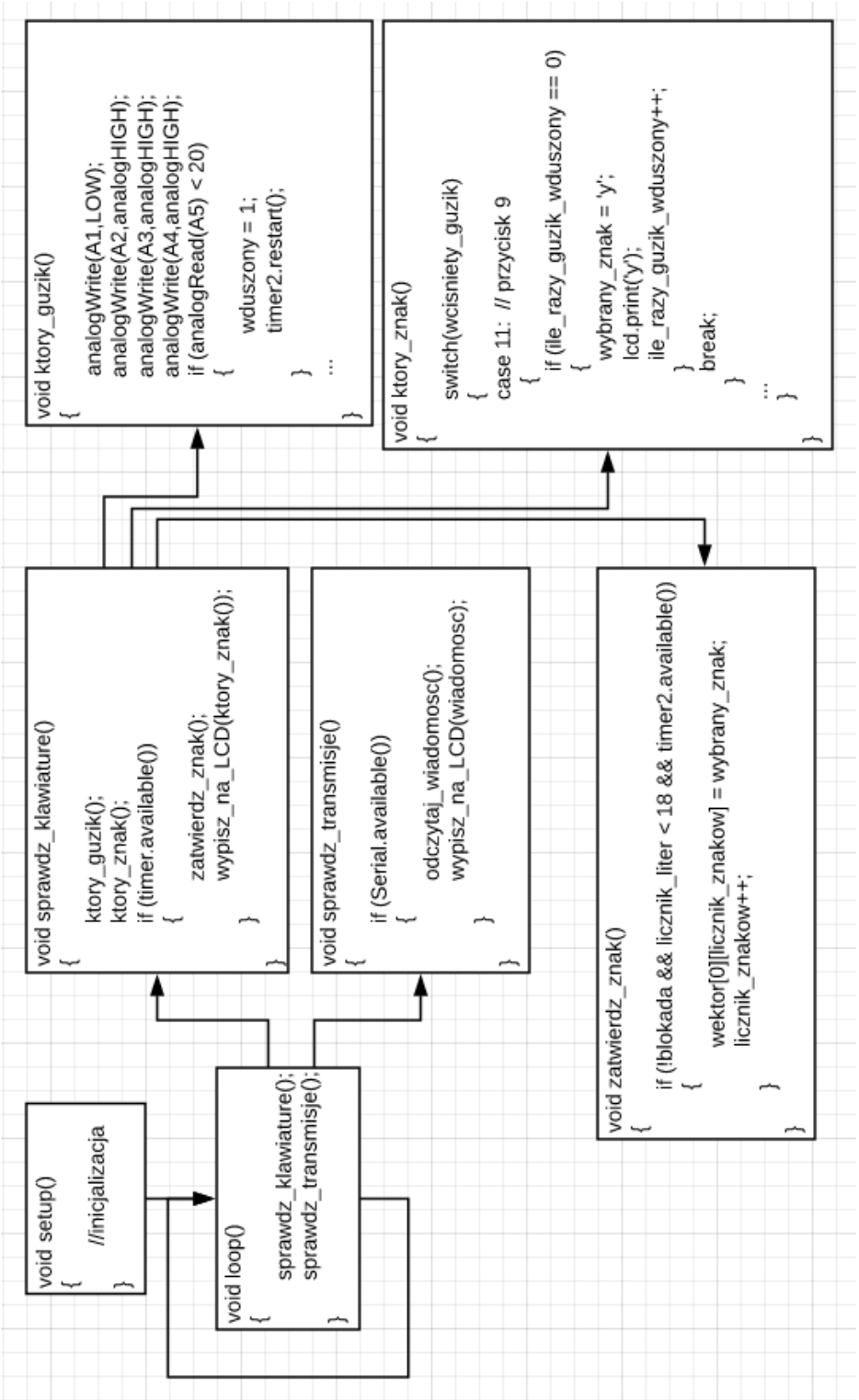
4.2. Organizacja oprogramowania

Głównym założeniem przy pisaniu programu jest programowanie blokowe. Polega ono na budowaniu funkcjonalnego kodu w fragmentach, które później można połączyć w jednolitą, współgrającą ze sobą całość.

Programowanie zostanie rozpoczęte od oprogramowania i sprawdzenia możliwości poszczególnych komponentów, tj. wyświetlacza, modułu radiowego, klawiatury. Umożliwi to wczesne odnalezienie błędów lub wad komponentów.

Środowisko programowania to wspomniane w rozdziale 1.1 Arduino IDE, które posiada mechanizm sprawdzania błędów w składni kodu oraz możliwość kompilacji kodu napisanego w języku wysokiego poziomu do kodu niskiego poziomu, który procesor rozumie i wczytuje przez odpowiednie złącze, w tym przypadku Mikro USB.

Schemat oprogramowania przedstawia rysunek 4.1.



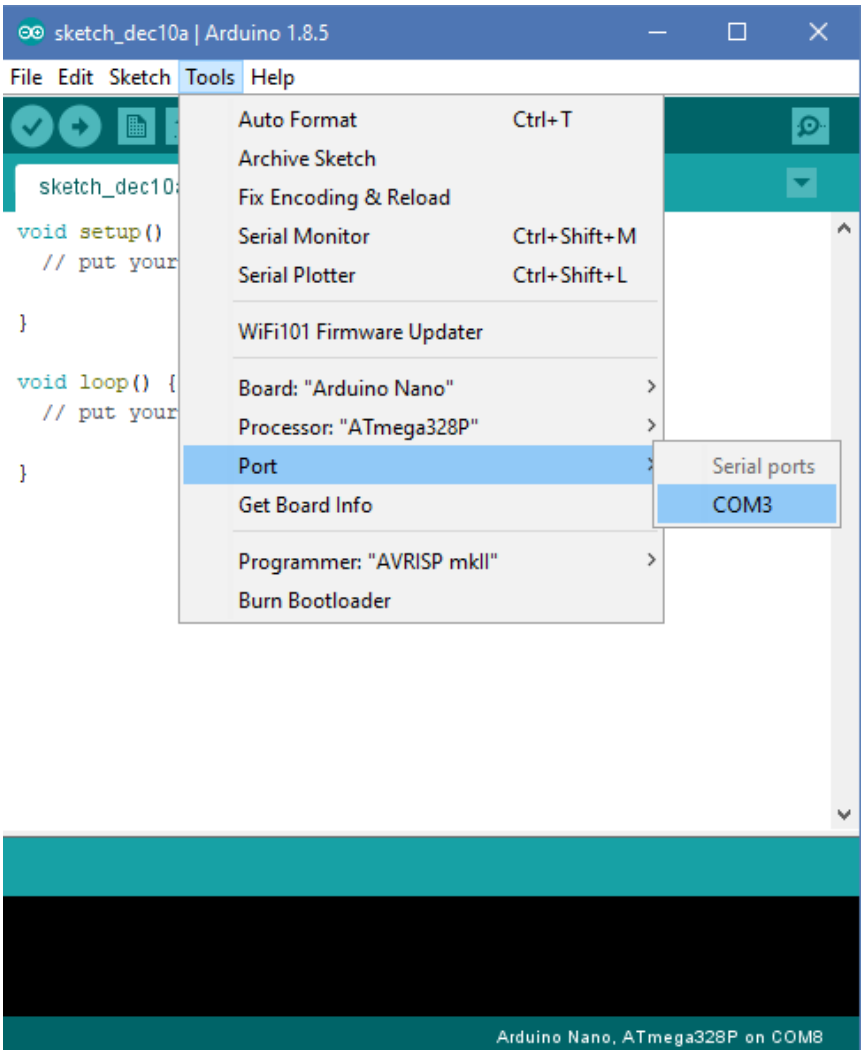
Rys. 4.1 Schemat ideowy programu

Arduino Nano posiada wbudowany programator CH340. Skutkuje to z punktu widzenia użytkownika prostym podłączeniem i wczytaniem kodu.

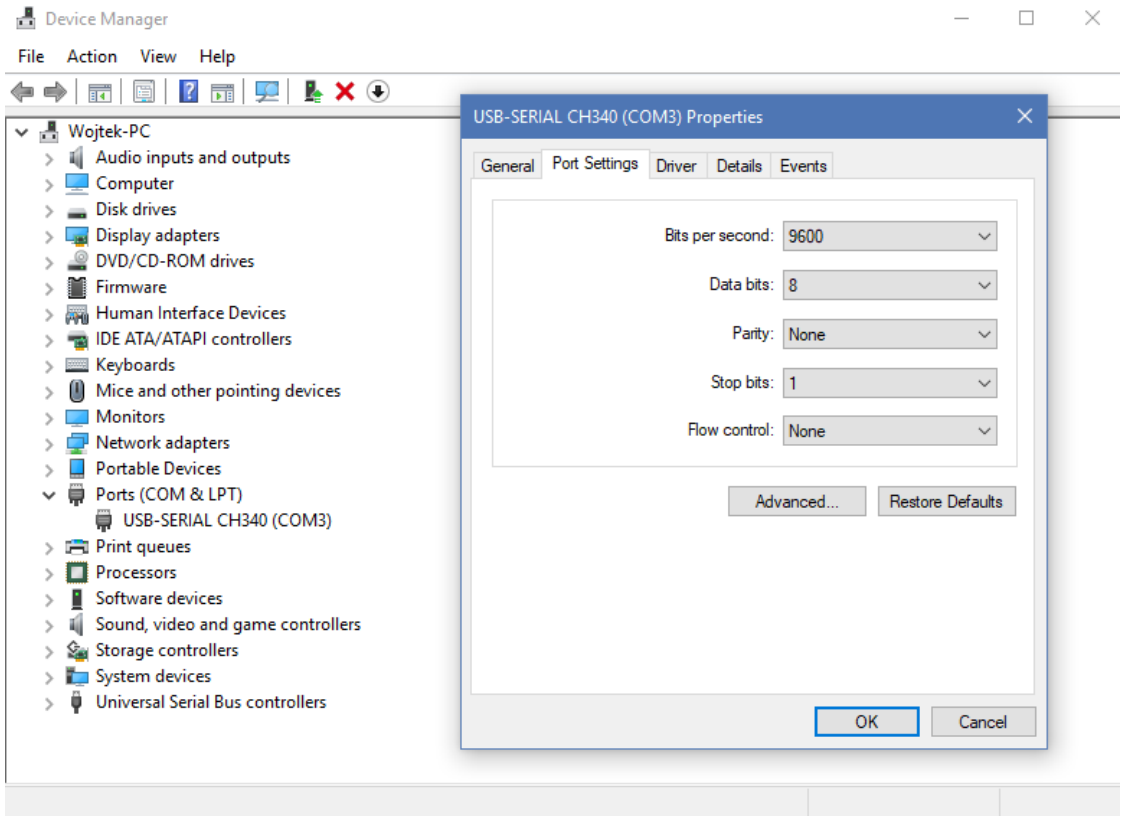
Aby móc przystąpić do wgrywania kodu po jego napisaniu, należy ustawić odpowiednią konfiguracje w ustawieniach środowiska (tabela 4.1, rys. 4.2). Należy też w systemie Windows skonfigurować port COMx, aby ustawienia (rys. 4.3) zgadzały się z kodem programu.

Tabela 4.1 Konfiguracja środowiska Arduino IDE

Pole	Wartość
Board	Arduino Nano
Processor	Atmega328p
Port	Odpowiednio podłączony COMx



Rys. 4.2 Konfiguracja środowiska Arduino IDE

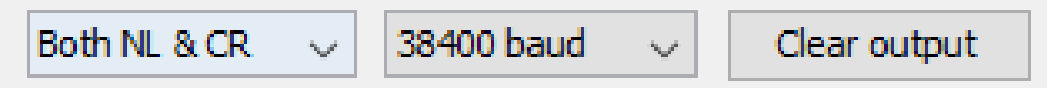


Rys. 4.3 Konfiguracja portu COM3

4.3. Obsługa modułu radiowego

Podstawowe informacje dotyczące organizacji programowania zostały przedstawione wcześniej. Po wykonaniu opisanych czynności mamy możliwość skonfigurowania modułu radiowego. W tym celu należy ustawić odpowiednie dla modułu parametry transmisji. W przypadku modułu HC-05 jest to prędkość transmisji 38400 oraz opcja zakończenia linii „BOTH CL & NR" (rys. 4.4).

Aby rozpocząć konfigurację pin EN należy połączyć z pinem 3V3 na mikrokontrolerze, a w chwili włączania zasilania przytrzymać przycisk obecny na module, który jest podłączony do omawianego pinu. Dioda zacznie mrugać z sekundową przerwą. Oznacza to, że moduł przeszedł w tryb AT, czyli konfiguracji.



Rys. 4.4 Ustawienia Serial Monitora

Po wykonaniu opisanych czynności do mikrokontrolera można wczytać kod przedstawiony, poniżej, który pozwala na wykorzystanie Serial Monitora do wysyłania i odbierania danych z UART.

Konfiguracja modułu radiowego

```
1     #include <SoftwareSerial.h>
2     SoftwareSerial mojSerial(3, 2); // RX, TX
3
4     String command = "";
5
6     void setup()
7     {
8           Serial.begin(38400);
9           Serial.println("AT");
10          mojSerial.begin(38400);
11    }
12
13    void loop()
14    {
15          if (mojSerial.available())
16          {
17                while(mojSerial.available())
18                      command += (char) mojSerial.read();
19          }
20          Serial.println(command);
21          command = "";
22          if (Serial.available())
23          {
24                delay(10);
25                mojSerial.write(Serial.read());
26          }
27    }
```

Aby nie kolidować ze sprzętowym UART i wgrywaniem kodu na płytkę użyty został zaprogramowany UART (pin 2 oraz 3 odpowiednio, jako RX i TX). Funkcjonuje on identycznie jak sprzętowy odpowiednik dzięki bibliotece SoftwareSerial, która jest jedną z dołączanych do środowiska. Aby uprościć konfigurację prędkość portu została ustawiona na identyczną z prędkością transmisji modułu radiowego (linia 8 i 10). Funkcja *void loop()* (linia 13 rys. 5.4) przedstawia sposób odczytu danych z Serial Monitora i jeśli nie ma aktualnie transmisji w obiekcie klasy SoftwareSerial mojSerial to dokonuje transmisji za jego pośrednictwem. W linii 23 rysunku 5.4 występuje element delay(10). Jest to wymagany element, aby ten sposób transmisji działał poprawnie. W linii 15 jest sprawdzenie, czy mojSerial odebrał wiadomość i jeśli tak, to ją odczytuje znak po znaku.

Komendy wymagane do skonfigurowania modułu są przedstawione w tabeli 4.2 w kolejności ich wprowadzania z podziałem na role urządzeń, czyli Slave i Master. Na początek w celu weryfikacji działania komunikacji wpisujemy testową wiadomość AT. Odpowiedzią na nią i na wszystkie pozostałe wiadomości, które odpowiadają potwierdzeniem jest OK.

W przedstawianym programie kod niezwiązany z wyświetlaniem tekstu na wyświetlaczu i inne funkcje, które nie są do transmisji potrzebne został pominięty.

Tabela 4.2 Zestaw komend AT

RTTmaster	RTTslave	Opis
AT	AT	Test komunikacji
AT+ADDR?	AT+ADDR?	Otrzymanie adresu modułu
AT+ORGL	AT+ORGL	Wyczyszczenie konfiguracji
AT+RMAAD	AT+RMAAD	Wyczyszczenie listy adresów
AT+UART=38400,1,0	AT+UART=38400,1,0	Ustaw. prędkości transmisji
AT+PSWD=1122	AT+PSWD=1122	Ustawienie hasła
AT+NAME=RTTmaster	AT+NAME=RTTslave	Nazwanie modułu
AT+ROLE=0	AT+ROLE=1	Ustawienie roli
AT+CMODE=0	AT+CMODE=0	Łączenie wg listy adresów
AT+INIT	AT+INIT	Inicjalizacja profilu SPP[1]
AT+LINK=98d3,31,fc4ad2	AT+LINK=98d3,32,30dd05	Dodanie do listy adresów

Odczytywanie danych z programowego portu szeregowego, nazwanego w programie głównym *mySerial*, jest zaimplementowane w funkcji *odczytaj_serial* (kod poniżej).

Odczyt z mySerial

```
1       void odczytaj_serial()
2       {
3         ...
4         while(mySerial.available() && i <= 20)
5         {
6           ostatnie_wiadomosci[0][i] = mySerial.read();
7           delay(30);
8           i++;
9         }
10        mySerial.read();
11      }
```

W linii 10 można zauważyć odczyt z mySerial, który występuje poza pętlą. Jest to zabezpieczenie, żeby zostało odczytane wszystko, ale niezapisane do tablicy. Ostatni znak pozostający w buforze to nie ostatni znak przesyłanej wiadomości tylko losowy znak. W celu jego pozbycia się, aby nie był wyświetlany w żaden sposób jest odczytywany poza główną petlą i nieuwzględniany przy dodawaniu znaku do tablicy. Funkcja opóźniająca 30 ms jest tutaj ponownie potrzebna w celu poprawnego działania procedury.

Wysyłanie zaimplementowane jest w funkcji *wybor_przycisku* w momencie wciśnięcia przycisku 4, czyli „A” na klawiaturze (kod pt. Wysyłanie wiadomości i jej czyszczenie po wysłaniu). Wysłana jest tablica znaków typu *char*.

[1] Serial Port Profile

Wysyłanie wiadomości i jej czyszczenie po wysłaniu

```
1       bool wybor_przycisku()
2       {
3         ...
4               switch(wduszony_guzik)
5               {
6                 case 4:  // przycisk A - wyslij
7                   {
8                     mySerial.write(wiadomosc_do_wyslania);
9                     dodaj_wyslana_wiadomosc();
10                    for (int i = 0;i < 18;i++)
11                    {
12                      wiadomosc_do_wyslania[i] = ' ';
13                    }
14                    ...
19                    break;
20                  }
21              ...
22              }
23       ...
24      }
```

Dodawanie kolejnych liter przedstawia poniższy kod.

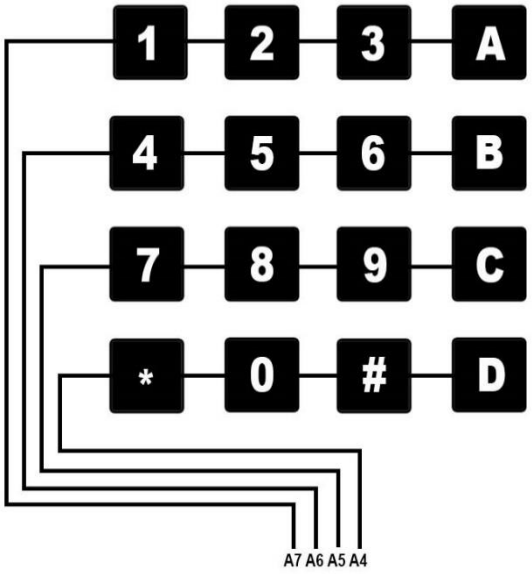
Dodawanie kolejnych liter

```
1       void zatwierdz_litere()
2       {
3         if (czas_wduszenia.available())
4         {
5           wybor_przycisku();
6           if (wybor_guzika.available() && !blokada && licznik_liter < 18)
7           {
8       ...
9               if (!usuwanie)
10              {
11                wiadomosc_do_wyslania[licznik_liter] = wybrana_litera;
12                licznik_liter++;
13                ...
14              }
15             ...
16    }
17          ...
18        }
19      }
```

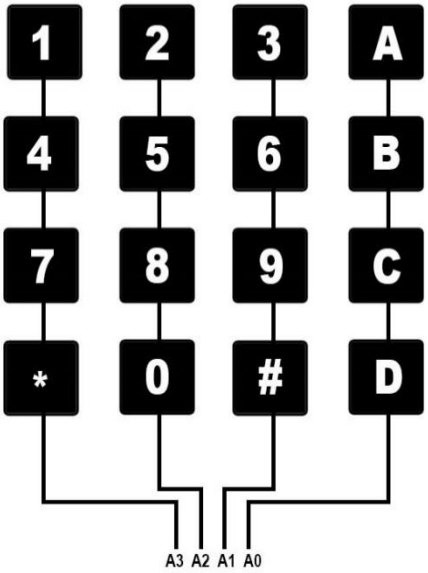
4.4. Obsługa klawiatury

Do projektu wybrano klawiaturę membranową typu matrycowego (rys. 4.5), której schemat przedstawiają rysunki 4.6a i 4.6b. Jest to prosta w budowie konstrukcja, polegająca na zwieraniu wejścia z wyjściem. Aby ją podłączyć wystarczy użyć 8 pinów mikrokontrolera lub użyć multiplexera w celu zmniejszenia liczby wykorzystywanych do tego celu pinów.

Rys. 4.5 Klawiatura membranowa, matrycowa [21]



Rys. 4.6a Schemat wejść klawiatury



Rys. 4.6b Schemat wyjść klawiatury

W programie do obsługi klawiatury wykorzystano 8 wolnych pinów. Zaletą jest ich ułożenie, które jest w jednej linii od A0 do A7 włącznie.

Atmega328p na pinach A7 i A6 nie posiada rezystorów, które zmienią potencjał pinu. Potrzebne, zatem jest ich dołączenie w projekcie. Wykorzystanie rezystorów jest istotne. Im większa rezystancja tym szybciej wartość napięcia z 5V wraca do 0V. Bez rezystorów piny zachowują się jak kondensatory i napięcie z nich spada bardzo powoli. Powoduje to problemy z wybieraniem przycisku, bo pojawia się sytuacja, gdzie po wciśnięciu przycisku połączonego z wierszem A7 utrzymuje się na nim jeszcze przez chwilę stan wysoki i mikrokontroler odbiera to, jako wciśnięcie przycisku.

Piny zdefiniowano według ich roli tj. A0 jako *input1*, A3 jako *input4*, A4 jako *output1*, A7 jako *output4* za pomocą definicji preprocesora.

Definicja wejść/wyjść klawiatury

```
1       #define input1 A0
2       #define input2 A1
3       #define input3 A2
4       #define input4 A3
5       #define output1 A4
6       #define output2 A5
7       #define output3 A6
8       #define output4 A7
```

Program do obsługi klawiatury został podzielony na dwie części. Pierwsza część, funkcja *sprawdz_klawiature* (kod poniżej) to obsługa mechaniczna klawiatury, czyli wystawianie na wejście jednej z linii wejściowych logicznego zera, gdy inne pozostają w stanie wysokim. Po wciśnięciu przycisku funkcja przesyła numer przycisku do funkcji *guzik_wduszony* z odpowiadającą mu wartością przycisku idąc od lewego, górnego rogu. W przeciwnym razie, jeśli żaden przycisk w danym momencie nie został wybrany, wartość zmiennej *wduszony* przyjmuje wartość *false* oraz funkcja zwraca 0.

Sprawdzanie wciśnięcia przycisków

```
1       bool sprawdz_klawiature()
2       {
3         for (linia = 1;linia < 5;linia++)
4         {
5           if (linia == 5)
6           {
7             linia = 1;
8           }
9
10          if (linia == 1)
11          {
12               // wybor lini
13              analogWrite(input1,LOW);
14              analogWrite(input2,analogHIGH);
15              analogWrite(input3,analogHIGH);
16              analogWrite(input4,analogHIGH);
17
18              if (analogRead(output1) < 20)
19              {
20                guzik_wduszony(1);
21                return 1;
22              }
23              else if (analogRead(output2) < 20)
24              {
25                guzik_wduszony(2);
26                return 1;
27              }
28              ...
29              }
30            }
31       ...
32        wduszony = false;
33        return 0;
34      }
```

Funkcja *guzik_wduszony* (kod poniżej stawowi zabezpieczenie. Sprawdzane jest tutaj czy nie został wybrany inny przycisk niż wybrany na wstępie oraz czy może żaden przycisk już nie jest wybierany. Jeśli oba warunki są spełnione, to flaga *wduszony* przyjmuje wartość *true* i można skorzystać z drugiej części obsługi klawiatury tj. programowego wyboru znaku. W funkcji resetowany jest licznik czasu przyciśnięcia, który sprawia, że przytrzymanie przycisku nie powoduje wybierania ponownie znaku, jest traktowane, jako jedno wciśnięcie.

Zabezpieczenie wciśnięcia przycisku

```
1     void guzik_wduszony(int ktory_guzik) // funkcja działania guzika
2     {
3       wduszony_guzik = ktory_guzik;
4       if (poprzedni_wduszony_guzik == wduszony_guzik && !wduszony)
5       {
6         wduszony = true;
7         poprzednia_linia = linia;
8       }
9       poprzedni_wduszony_guzik = wduszony_guzik;
10      czas_wduszenia.restart();
11    }
```

Druga część obsługi klawiatury, to część wybierania znaku dla odpowiedniej liczby wciśnięć przycisku. Odpowiada za to funkcja *wybor_przycisku* (kod na następnej stronie), która w warunkach wywołania dalszego kodu wywołuje i sprawdza, co zwraca funkcja *sprawdz_klawiature*. Jeśli wartość *sprawdz_klawiature* jest prawdziwa to znaczy, że jest wciśnięty przycisk. Sprawdzane jest także, czy nie jest wciśnięty żaden inny przycisk. Wywoływana jest funkcja generująca dźwięk wciśnięcia przycisku *dzwiek_klawiatury*.

W funkcji *wybor_przycisku* tworzymy podział na przyciski, które działają zawsze niezależnie od tego, co się dzieje na ekranie i na te, które mogą zostać zablokowane w pewnych warunkach, głównie są to przyciski wyboru znaku. Przyciski wyboru przycisku są blokowane w chwili osiągnięcia limitu znaków wraz z odpowiednią informacją.

Flaga *wybierana_litera* jest ustawiana w chwili, gdy chcemy wybrać znak ponad pojemność znakową wyświetlacza, czyli 20 minus 2 tj. 18 liter.

W chwili, gdy wybierzemy przycisk 13, czyli „*” włączymy duże litery i flaga *duze_litery* przyjmie wartość *true*, wyświetli się też adekwatna informacja. Po pownonym wybraniu gwiazdki zostają wybrane małe litery.

Funkcja *informacja* służy do wyświetlania informacji. „Limit znaków” wyświetlany jest dla argumentu 1. Dla innych wiadomości typu „abc/ABC”, „Connected” itd. należy podać inne argumenty odpowiednio wybierające dany typu wybieranej wiadomości. Trwa ona w zależności o rodzaju od pół sekundy do trzech sekund.

Schemat przypisania znaków do przycisków przedstawiony jest w dodatku C.

Wybór znaku

```
1       void wybor_przycisku()
2       {
3           if (sprawdz_klawiature() && !wduszony)
4           {
5             dzwiek_klawiatury();
6             ...
7             if (!wybierana_litera)
8             {
9               switch(wduszony_guzik)
10              {
11                case 4:  // przycisk A - wyslij
12                  {
13                    ...
14                    break;
15                  }
16                  case 3:  // przycisk B
17                  {
18                    ...
19                    break;
20                  }
21             ...
22              }
23            }
24
25            wybierana_litera = true;
26            if (!duze_litery)
27            {
28                     if (licznik_liter < 18)
29                     {
30                           switch(wduszony_guzik)
31                     {
32                      ...
33                            case 6:  // przycisk 9
34                            {
35                                  if (ile_razy_guzik_wduszony == 0)
36                                  {
37                                    wybrana_litera = 'y';
38                                    DEBUG_PRINT('y');
39                                    lcd.print('y');
40                                    ile_razy_guzik_wduszony++;
41                                    wybor_guzika.restart();
42                                  }
43                                  else if (ile_razy_guzik_wduszony == 1)
44                                  {
45                                    ...
46                                  }
47                                  ...
48                                  break;
49                            }
50                     ...
51                      }
52              }
53              else
54               informacja(1);
55             }
56      ...
57      }
```

4.5. Obsługa wyświetlacza

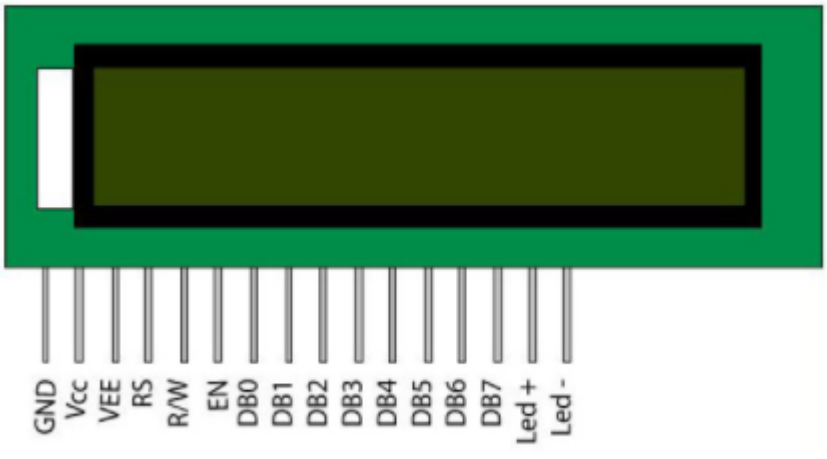
Wykorzystany w projekcie wyświetlacz to alfanumeryczny ciekłokrystaliczny wyświetlacz z podświetleniem ledowym. Do projektu został wybrany z powodu dużego wsparcia dla mikrokontrolerów z rodziny AVR oraz łatwości obsługi. Wybrany został model z 20 znakami w poziomie i 4 wierszami, w skrócie 4x20 oraz z podwójnym złączem połączeniowym. Podświetlenie jest koloru niebieskiego, a znaki są koloru białego. Sterownik jest zgodny z HD44780. Posiada on szeroką gamę znaków (rys. 4.7) do wyboru, niestety brak jest znaków polskich.

Rys. 4.7 Tablica znaków [22]

Do obsługi wyświetlacza użyto jednej z podstawowych bibliotek dołączonych wraz z oprogramowaniem, *LiquidCrystal*. Wykaz pinów przedstawia rysunek 4.8, a ich funkcje rysunek 4.9. W chwili tworzenia obiektu klasy *LiquidCrystal* jako argumenty musimy podać numery pinów użytych do podłączenia odpowiednich wejść.

W projekcie wykorzystano 4 linie danych, w celu zmniejszenia liczby wyprowadzeń. Pozwala na to sterownik HD44780. Pin VEE, regulujący kontrast został na stałe podłączony do VCC.

W funkcji setup, która zawiera instrukcje startowe, które wykonują się po uruchomieniu mikrokontrolera, musimy uruchomić wyświetlacz metodą begin, podając w argumencie liczbę kolumn i wierszy wyświetlacza „lcd.begin(20, 4);”, gdzie lcd to obiekt klasy *LiquidCrystal*.



Rys. 4.8 LCD i jego wyprowadzenia [23]

1	VSS	GROUND	0V (GND)
2	VCC	POWER SUPPLY FOR LOGIC CIRCUIT	+5V
3	VEE	LCD CONTRAST ADJUSTMENT	
4	RS	INSTRUCTION/DATA REGISTER SELECTION	RS = 0 : INSTRUCTION REGISTER RS = 1 : DATA REGISTER
5	R/W	READ/WRITE SELECTION	R/W = 0 : REGISTER WRITE R/W = 1 : REGISTER READ
6	E	ENABLE SIGNAL	
7	DB0	DATA BUS	8 BIT: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	LED+	SUPPLY VOLTAGE FOR LED+	+5V
16	LED-	SUPPLY VOLTAGE FOR LED-	0V

Rys. 4.9 Opis wyprowadzeń wyświetlacza LCD 4x20 [22]

Na przykładzie kodu z funkcji setup widać, że obsługa sprowadza się do użycia kilku podstawowych metod obiektu *lcd*. Wyświetlany jest tutaj tekst, który pojawi się tylko w chwili uruchomienia urządzenia. Biblioteka *LiquidCrystal* posiada kilka dodatkowych funkcjonalności, które w projekcie nie mają znaczenia. Projekt jednak wymaga, aby, kilka dodatkowych funkcjonalności niezawartych w bibliotece pojawiło się. Takie dodatkowe funkcjonalności to w tym przypadku blokada na końcu wiersza, aby kursor nie przeskakiwał na kolejną linię.

Wyświetlacz funkcją lcd.write jest w stanie przesyłać tylko tablice znaków. Dlatego wszystkie wiadomości są przechowywane w wektorach o wielkości 18.

Funkcja setup

```
1       void setup()
2       {
3         ...
4         lcd.begin(20, 4);
5         lcd.setCursor(0,0);
6         lcd.write("RADIO        RTT v.1");
7         lcd.setCursor(0,1);
8         lcd.write("TEXT");
9         lcd.setCursor(0,2);
10        lcd.write("TRANSMITER");
11        lcd.setCursor(0,3);
12        lcd.write("Wojciech Olszewski");
13        delay(3000);
14        lcd.clear();
15        lcd.blink();
16        ...
17      }
```

Blokadę kursora przedstawia kod poniżej. W chwili osiągnięcia limitu znaków, kursor pozostaje na ostatnim miejscu i litera się nie zatwierdzi. Jeśli natomiast zostanie zatwierdzona ostatnia litera, to kursor nie zmieni położenia na kolejne miejsce.

Ustawienie limitu znaków

```
1       void zatwierdz_litere()
2       {
3         if (wybor_guzika.available() && !blokada && licznik_liter < 18)
4         ...
5               if (licznik_liter < 18)
6               {
7               lcd.setCursor(licznik_liter+2,3);
8                pozycja_kursora = licznik_liter+2;
10              }
11              else
12              {
13              lcd.setCursor(licznik_liter+1,3);
14               pozycja_kursora = licznik_liter+1;
15              }
16       ...
17      }
```

4.6. Funkcje urządzenia

4.6.1. Wyświetlanie ostatnich wiadomości

Funkcja ta została zaimplementowana dla zachowania przejrzystości interfejsu. Logicznie informacje układane są na stos. Nowe są nakładane na stare itd. W programie należy wiadomości przechowywać w tablicach typu char. Do tego celu wykorzystywane są 3 tablice: 18 elementowa *wiadomość_do_wyslania* oraz macierz 2x21 *ostatnie_wiadomosci*. Macierz *ostatnie_wiadomosci* ma 20 znaków + 1 dodatkowy, dla losowego znaku. Pierwsze 2 to znaki interfejsu „R>” oraz „S>”, pozostałe 18 dla wiadomości. Funkcje przedstawia kod poniżej.

W programie została funkcja ta została nazwana *wypisz_ostatnie_wiadomosci*. Uwzględnia ona także, czy otrzymana lub wysłana wiadomość jest pierwsza i odpowiednio wyświetla wiadomości.

Funkcja wypisz_ostatnie_wiadomosci

```
1       void wypisz_ostatnie_wiadomosci(bool wyslana_czy_odebrana)
2       {
3         wiadomosc = true;
4         lcd.setCursor(0,0);
5         if (wyslana_czy_odebrana)
6         {
7           ostatnie_wiadomosci[0][0] = 'S';
8           DEBUG_PRINTLN("Wiadomosc wyslana");
9         }
10        else
11        {
12          ostatnie_wiadomosci[0][0] = 'R';
13          DEBUG_PRINTLN("Wiadomosc odebrana");
14        }
15        if (flaga_1_wiadomosci)
16          {
17            DEBUG_PRINTLN("Nie pierwsza wiadomosc");
18            if (!wyslana_czy_odebrana)
19            {
20              odczytaj_serial(); // funkcja odczytujaca znaki z seriala
21            }
22            ostatnie_wiadomosci[0][1] = '>';
23            DEBUG_PRINT("Zawartosc 1 lini: \"");
24            for (int i = 0;i < 20;i++)
25            {
26              DEBUG_PRINT(ostatnie_wiadomosci[0][i]);
27              lcd.write(ostatnie_wiadomosci[0][i]);
28            }
29            DEBUG_PRINTLN("\"");
30            lcd.setCursor(0,1);
31            ostatnie_wiadomosci[1][1] = '>';
32            DEBUG_PRINT("Zawartosc 2 lini: \"");
33            for (int i = 0;i < 20;i++)
34            {
35              DEBUG_PRINT(ostatnie_wiadomosci[1][i]);
36              lcd.write(ostatnie_wiadomosci[1][i]);
37            }
38            DEBUG_PRINTLN("\"");
39          }
40          else
41          {
42            DEBUG_PRINTLN("Pierwsza wiadomosc");
43
44            if (!wyslana_czy_odebrana)
45               odczytaj_serial(); // funkcja odczytujaca znaki z seriala
46
47            lcd.setCursor(0,0);
48            ostatnie_wiadomosci[0][1] = '>';
49
50            DEBUG_PRINT("Zawartosc 1 lini: \"");
51            for (int i = 0;i < 20;i++)
52            {
53              DEBUG_PRINT(ostatnie_wiadomosci[0][i]);
54              lcd.write(ostatnie_wiadomosci[0][i]);
55            }
56            DEBUG_PRINTLN("\"");
57
58            flaga_1_wiadomosci = true;
59          }
60      }
```

4.6.2. Czyszczenie ekranu ostatnich wiadomości

Funkcja w programie nazywa się *wyczyść_wektory* i ma za zadanie wyczyścić ekran ze znaków i przywrócić stan po uruchomieniowy. Oznacza to, że należy wyczyścić wektory przetrzymujące znaki oraz ustawić odpowiednie flagi. Flaga *wiadomosc* mówi nam o tym, czy wektory przechowują jakiekolwiek znaki (wykorzystywana jest do wyświetlania wiadomości „No messages" widocznej w przypadku braku jakichkolwiek wiadomości do wyświetlenia), a flaga *flaga_1_wiadomosci*, czy została już wyświetlona pierwsza wiadomość.

Funkcja wypisz_ostatnie_wiadomosci

```
1     void wyczysc_wektory()
2     {
3       lcd.clear();
4       lcd.setCursor(0,0);
5       lcd.print("No messages");
6       for (int i = 0;i < 21;i++)
7       {
8         ostatnie_wiadomosci[0][i] = ' ';
9         ostatnie_wiadomosci[1][i] = ' ';
10      }
11      for (int i = 0;i < 18;i++)
12      {
13        wiadomosc_do_wyslania[i] = ' ';
14      }
15      rysuj_interfejs();
16      flaga_1_wiadomosci = false;
17      wiadomosc = false;
18    }
```

4.6.3. Przychodząca wiadomość podczas pisania

W trakcie pisania i testowania przesyłania wiadomości odkryty został problem, który nie był prosty w rozwiązaniu. Dotyczył on tego, jak wyświetlić wiadomość, która jest odbierana w chwili pisania i wybierania literki. Dochodziło tutaj do przemieszczania się kursora i wpisywania znaku w losowym miejscu. Z początku rozwiązaniem była blokada odczytania wiadomości z bufora do momentu wysłania pisanej wiadomości lub do momentu wyczyszczenia jej.

W trakcie pisania dalszej części programu, pojawił się pomysł, aby w trakcie wybierania literki Arduino sprawdzało kilkaset razy, czy coś się pojawia na wejściu programowego UART. Jeśli wykryje nadaną wiadomość, ustawia blokadę flagą *blokada* i zostaje wstrzymana możliwość wyboru literki. Po odebraniu i odczytaniu wiadomości flaga jest negowana i następuje wznowienie możliwości pisania.

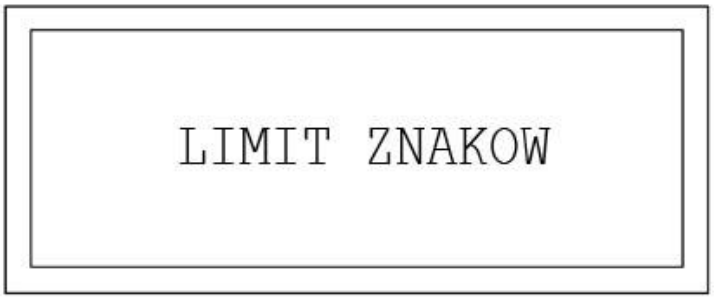
Wyświetlenie zawartości wiadomości zajmuje blisko 1 sekundę. Dlatego sztuczka z flaga *blokada* pozostaje niemal niezauważalna dla użytkownika.

4.6.4. Wiadomości informacyjne

Wiadomość informacyjną można wywołać funkcją *informacja*(*argument*), gdzie argument to numer informacji (kod poniżej). Każda informacja zostaje wyświetlona na pustym tle oraz na środku ekranu przez określony czas (rys. 4.10). W chwili zakończenia wyświetlania, interfejs główny zostaje ponownie wyświetlony wraz z wiadomościami zapisanymi w macierzy *ostatnie_wiadomosci*, to znaczy dwoma ostatnimi wiadomościami.

Funkcja informacja

```
1     void informacja(int ktora_informacja)
2     {
3       DEBUG_PRINTLN();
4       lcd.setCursor(0,0);
5       lcd.write("                    ");
6       lcd.setCursor(0,2);
7       lcd.write("                    ");
8       lcd.setCursor(0,3);
9       lcd.write("                    ");
10      switch(ktora_informacja)
11      {
12        case 1:
13        {
14          digitalWrite(brzeczyk,LOW);
15          delay(50);
16          digitalWrite(brzeczyk,HIGH);
17          delay(50);
18          digitalWrite(brzeczyk,LOW);
19          delay(50);
20          digitalWrite(brzeczyk,HIGH);
21          DEBUG_PRINT("[LIMIT ZNAKOW]");
22          lcd.setCursor(0,1);
23          lcd.write("    LIMIT ZNAKOW    ");
24          delay(500);
25          lcd.clear();
26          rysuj_interfejs();
27          lcd.setCursor(2,3);
28          lcd.write(wiadomosc_do_wyslania);
29          wypisz();
30          lcd.setCursor(pozycja_kursora,3);
31          break;
32        }
33    ...
34    }
```



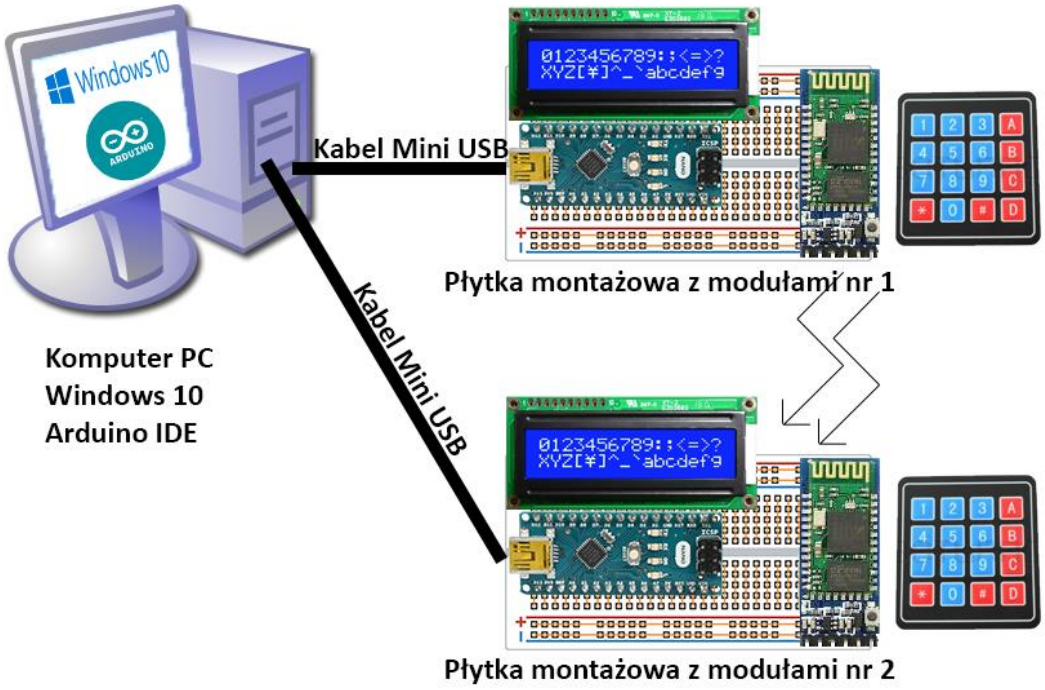
Rys. 4.10 Interfejs wiadomości informacyjnej

5. Uruchamianie systemu

5.1. Środowisko uruchomieniowe

Środowisko uruchomieniowe dla projektu to system operacyjny *Windows 10* w wersji Professional oraz program *Arduino*. Oba elementu środowiska są niezbędne do wczytywania kodu do mikrokontrolera i jego debugowania w razie błędów.

Kolejnymi elementami środowiska uruchomieniowego są płytka stykowa inaczej zwana prototypową lub montażową, na której łączone są moduły. Ostatnimi elementami są same moduły. Środowisko przedstawia rysunek 5.1.



Rys. 5.1 Schemat środowiska uruchomieniowego

5.2. Testy komponentów na płytkach montażowych

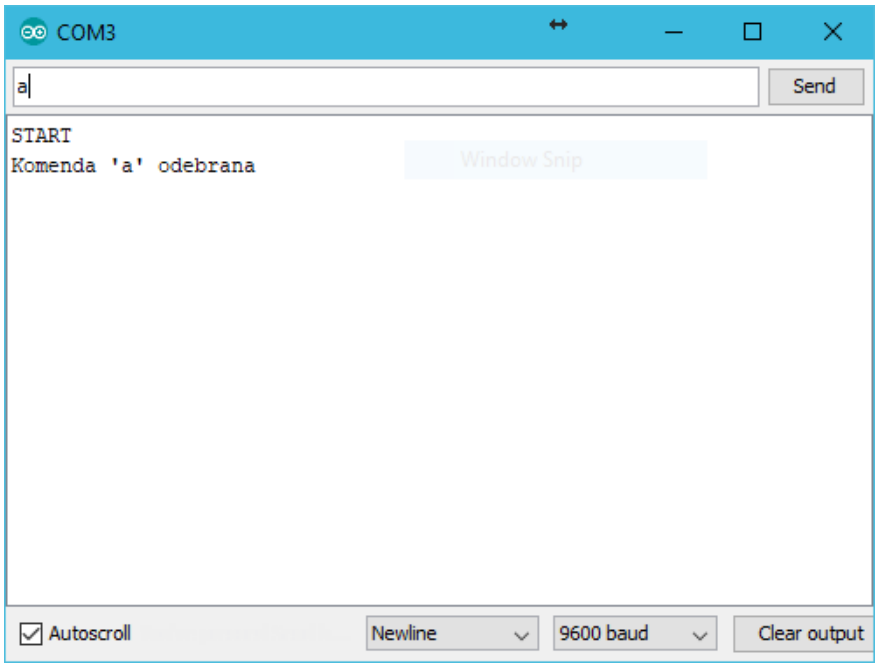
5.2.1. Testy komunikacji mikrokontrolera z komputerem

Pierwszym testem jest test komunikacji szeregowej. Komunikacja szeregowa jest potrzebna do wczytywania kodu do mikrokontrolera oraz do odbierania wiadomości zwrotnych zdefiniowanych przez programistę. Zazwyczaj wykorzystuje się je w celu sprawdzania błędów tzw. „debugowania" lub sprawdzania stanu urządzenia.

Test polega na wczytaniu kodu (przykładowy kod poniżej) poprzez program *Arduino IDE* do mikrokontrolera za pomocą kabla Mini USB. Program pozwala na wpisywanie z klawiatury wiadomości poprzez *Serial Monitor*. Jeśli mikrokontroler rozpozna komendę, odeśle adekwatną wiadomość także poprzez *Serial Monitor* (rys. 5.2).

Przykładowy kod programu sprawdzania komunikacji

```
1       void setup()
2       {
3         Serial.begin(9600);
4         Serial.write("START");
5       }
6       void loop()
7       {
8         if (Serial.available())
9         {
10          if (Serial.read() == 'a')
11          {
12            Serial.write("\nKomenda 'a' odebrana");
13          }
14        }
15      }
```



Rys 5.2 Serial Monitor po odebraniu komendy

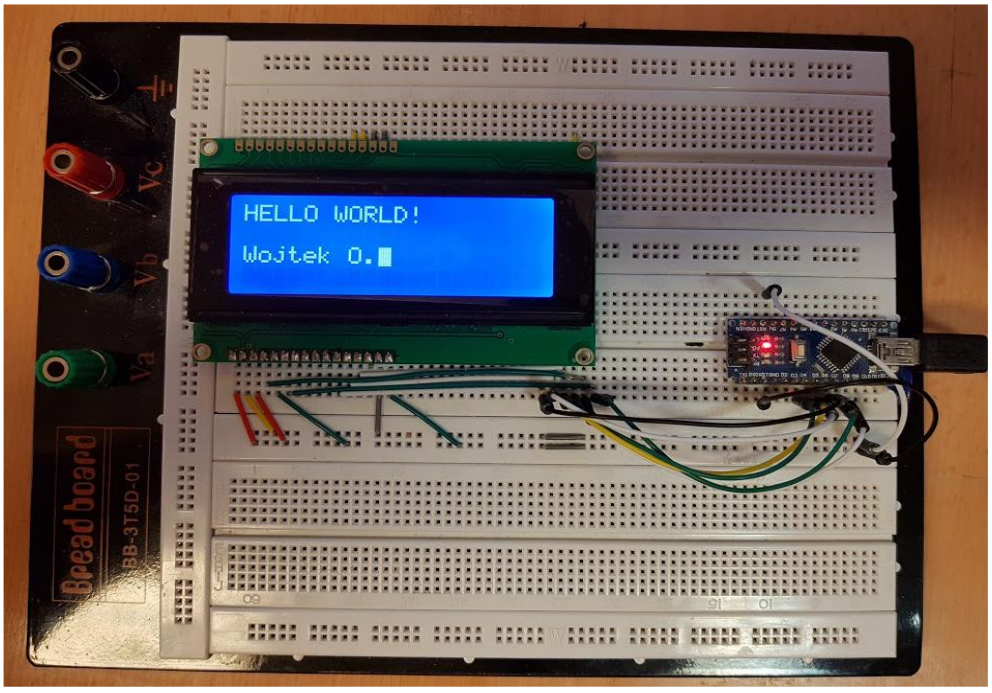
5.2.2. Testy funkcji wyświetlacza

Test funkcji wyświetlacza polega na sprawdzeniu komunikacji wyświetlacza z mikrokontrolerem oraz sprawdzeniu działania poszczególnych wymaganych w projekcie funkcji.

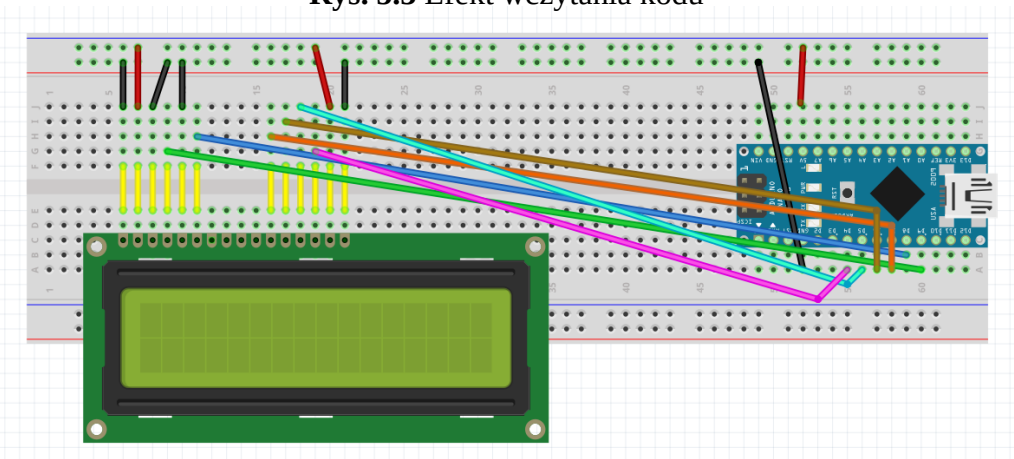
Obsługa wyświetlacza sprowadza się do użycia metod klasy *LiquidCrystal* (kod poniżej). Klasa ta służy do obsługi wyświetlaczy zgodnych ze sterownikiem HD44780. Więcej informacji na ten temat zamieszczone w rozdziale 4.5. Efekt wczytania kodu do mikrokontrolera ilustruje rysunek 5.3. Rysunek 5.4 przedstawia natomiast sposób, w jaki wyświetlacz został podłączony.

Kod programu obsługi wyświetlacza

```
1       #include <LiquidCrystal.h>
2
3       LiquidCrystal LCD(9, 8, 7, 6, 5, 4); // RS,EN,DB4,DB5,DB6,DB7
4
5       void setup()
6       {
7         LCD.begin(20, 4);
8         LCD.print("HELLO WORLD!");
9         LCD.setCursor(0,2);
10        LCD.write("Wojtek O.");
11        LCD.blink();
12      }
13
14      void loop()
15      {}
```



Rys. 5.3 Efekt wczytania kodu



Rys. 5.4 Sposób podłączenia LCD

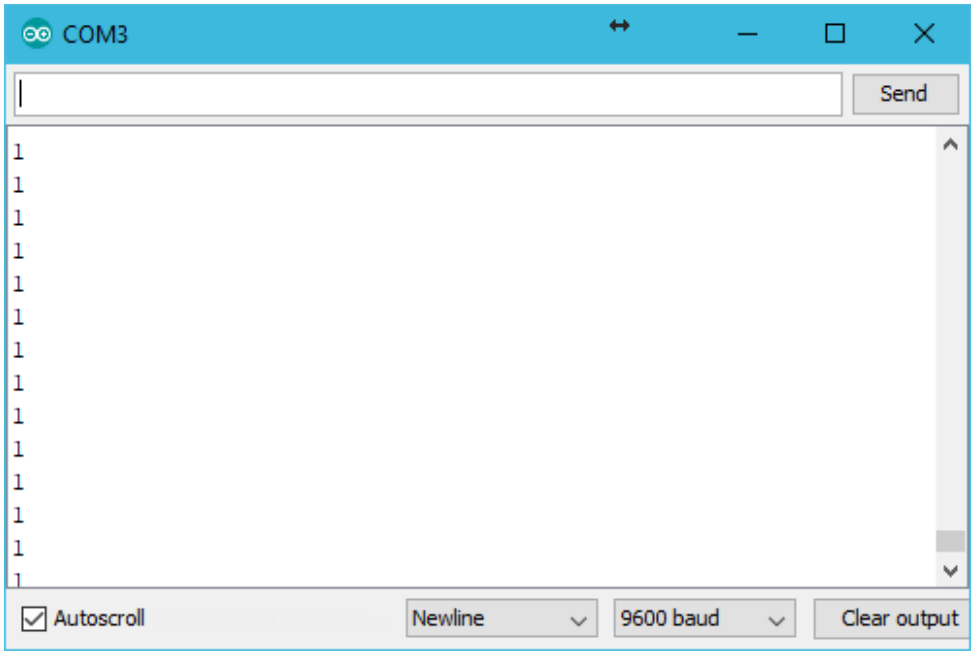
5.2.3. Testy klawiatury

Test klawiatury polega na zwieraniu poszczególnych pinów tak, aby napięcie z wejścia pojawiało się na wyjściu. Piny wejściowe nazwane *inputX* (X to kolejne cyfry) muszą być dołączone do wejść mikrokontrolera za pomocą rezystorów, aby potencjał zmieniał się wraz ze zwieraniem pinów.

Program testowy (kod poniżej) został napisany tak, aby wyświetlał numer przycisku w zależności od numeru wiersza i kolumny (rys. 5.5). Zwarcie A0 z A4 tj, przycisku „1" powoduje odpowiedź programu w postaci „1", czyli przycisk pierwszy. Program nie zabezpiecza przed drganiem styków, tylko testuje zwieranie linii.

Kod programu obsługi przycisków

```
1       #define input1 A0
2       #define input2 A1
3       #define input3 A2
4       #define input4 A3
5       #define output1 A4
6       #define output2 A5
7       #define output3 A6
8       #define output4 A7
9       #define analogHIGH 1024
10
11      void setup()
12      {
13        Serial.begin(9600);
14        pinMode(input1,OUTPUT);
15        pinMode(input2,OUTPUT);
16        pinMode(input3,OUTPUT);
17        pinMode(input4,OUTPUT);
18        digitalWrite(input1,HIGH);
19        digitalWrite(input2,HIGH);
20        digitalWrite(input3,HIGH);
21        digitalWrite(input4,HIGH);
22        pinMode(output1,INPUT_PULLUP);
23        pinMode(output2,INPUT_PULLUP);
24      }
25
26      void loop()
27      {
28        analogWrite(input1,LOW);
29        analogWrite(input2,analogHIGH);
30        analogWrite(input3,analogHIGH);
31        analogWrite(input4,analogHIGH);
32        if (analogRead(output1) < 20)
33        {
34          Serial.println('1');
35        }
36        else if (analogRead(output2) < 20)
37        {
38           Serial.println('2');
39        }
40        else if (analogRead(output3) < 20)
41        {
42          Serial.println('3');
43        }
44        else if (analogRead(output4) < 20)
45        {
46          Serial.println('4');
47        }
48      }
```



Rys. 5.5 Widok z Serial Monitor

5.2.4. Testy modułu radiowego

Test modułu radiowego polega na przesłaniu krótkiego tekstu przez protokół Bluetooth. W programie wpisywana jest na sztywno wiadomość, które ma zostać przesłana. Jak się okazuje, obiekt klasy Serial nie pozwala na przesłanie obiektu typu String. Wymagane jest, aby przesłana została zmienna typu char lub tablica typu char. Po zmodyfikowaniu programu i zadeklarowaniu tablicy stałej długości typu *char* wiadomość jest przesyłana. Ogranicza nam to jednak długość wiadomości do jednej z góry znanej długości. Ograniczenie to jest spowodowane organicznymi możliwościami klasy Serial oraz brakiem tworzenia dynamicznej tablicy w języku C.

Przed przystąpieniem do programowania oba moduły radiowe muszą być odpowiednio skonfigurowane według informacji zawartych w rozdziale 4.3.

Program do przesyłania wiadomości przedstawiony jest poniżej.

Program wysyłający przez Bluetooth wiadomość

```
1       #include <SoftwareSerial.h>
2       SoftwareSerial mySerial(3, 2); // RX,
TX
3
4       void setup()
5       {
6         Serial.begin(38400);
7         mySerial.begin(38400);
8       }
9       void loop()
10      {
11        if (Serial.available())
12        {
13             mySerial.write(Serial.read());
14        }
15      }
```

Wykorzystano tutaj *Serial Monitor*, do którego wpisywany jest tekst. Program pobiera z niego tekst i wysyła przez moduł Bluetooth do drugiego urządzenia.

Aby móc wiadomość odebrać należy zbudować lustrzane urządzenie z kodem, który będzie pozwalał na odebranie wiadomości i jej sygnalizację. W naszym przypadku wykorzystano wyświetlacz.

Przy testach drugie urządzenie (kod poniżej) działa na innej zasadzie. Sprawdza, czy moduł coś odebrał i wczytuje poszczególne litery do tablicy. Następnie całą tablicę wyświetla na wyświetlaczu.

Program odbierający tekst z modułu Bluetooth

```
1       #include <SoftwareSerial.h>
2       SoftwareSerial mySerial(3, 2); // RX, TX
3       #include <LiquidCrystal.h>
4       LiquidCrystal lcd(8, 9, 7, 6, 5, 4);
5       void setup()
6       {
7         Serial.begin(38400);
8         lcd.begin(20, 4);
9         mySerial.begin(38400);
10      }
11
12      char tablica[20] = {'R','>',' ',' ',' ',' ',' ','
',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '};
13      void loop()
14      {
15        lcd.setCursor(0, 0);
16        int i = 2;
17        if (mySerial.available())
18        {
19          lcd.clear();
20          while(mySerial.available() && i <= 20)
21          {
22            tablica[i] = mySerial.read();
23            delay(50);
24            i++;
25          }
26          lcd.write(tablica);
27        }
28      }
```

Efekt wpisania komendy „KOMENDA” w *Serial Monitor* przedstawia rysunek 5.6.

Rys. 5.6 Efekt wpisania w pierwszym urządzeniu komendy „KOMENDA”

5.3. Testy w systemie docelowym

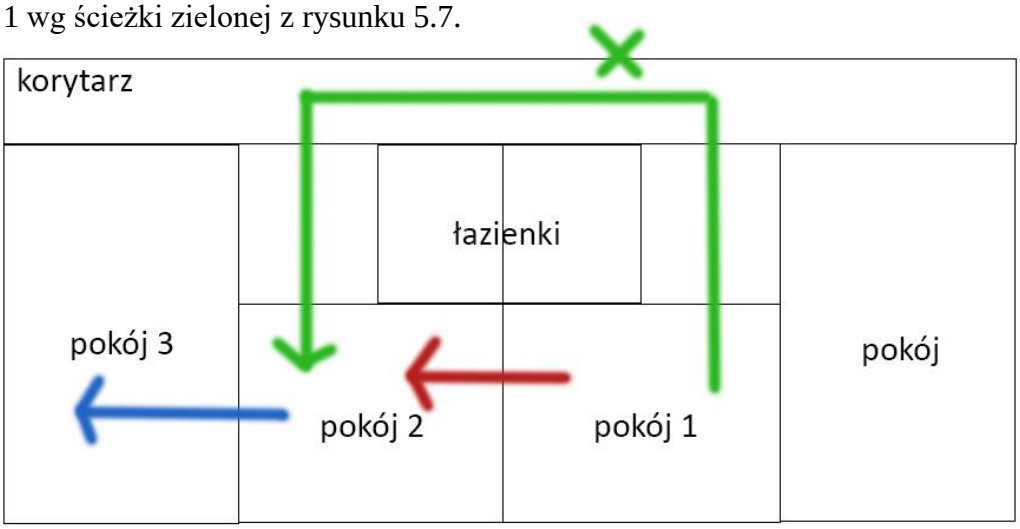
Pierwszym głównym testem, jaki należało wykonać to test zasięgu. Polegał on na tym, żeby oba urządzenia maksymalnie oddalać od siebie oraz jednocześnie wysyłać wiadomości w celu sprawdzenia ich poprawności.

W programie zaimplementowano zabezpieczenie przed utratą połączenia (kod, poniżej), które kontroluje stan pinu STATE. Jeśli pin STATE przyjmie wartość logicznego 0, to następuje utrata połączenia i na urządzeniu pojawi się informacja „CONNECTION LOST”. Wiadomość pokaże się także w chwili włączenia tylko jednego urządzenia i będzie widoczna, dopóki drugie urządzenie nie zostanie włączone.

Kod programu do wyświetlania informacji o braku połączenia

```
1       void sprawdz_polaczenie()
2       {
3         stan_polaczenia = digitalRead(connection);
4         if (stan_polaczenia == poprzedni_stan_polaczenia &&
stan_polaczenia == 1 && !potwierdzenie_polaczenia)
5         {
6           informacja(2);
7           potwierdzenie_polaczenia = true;
8           czas_od_polaczenia = millis();
9         }
10        else if (stan_polaczenia == poprzedni_stan_polaczenia
&& stan_polaczenia == 0)
11        {
12          informacja(3);
13          czas_od_polaczenia = 0;
14          delay(500);
15        }
16        else if (stan_polaczenia != poprzedni_stan_polaczenia
&& stan_polaczenia == 1 && poprzedni_stan_polaczenia == 0)
17        {
18          potwierdzenie_polaczenia = false;
19        }
20        poprzedni_stan_polaczenia = stan_polaczenia;
21      }
```

Pierwsze środowisko testowe to pokoje w akademiku nr 6. Test polegał na oddalaniu się z pokoju 1 wg ścieżki zielonej z rysunku 5.7.



Rys. 5.7 Schemat pokojów akademika oraz ścieżki testowania

Punkt X na ścieżce zielonej oznacza utratę zasięgu, który został odzyskany przed pokojem 2. Odzyskany zasięg oznacza strzałka czerwona. Wiadomości, które zostały wysłane w chwili utraty zasięgu zostały odebrane w drugim urządzeniu po zniknięciu wiadomości informacyjnej o utracie zasięgu. Testowano także przesył wiadomości tylko w kierunku czerwonej strzałki i wiadomości przechodziły przez grubą ścianę bez problemu. Test został wykonany także w kierunku niebieskiej strzałki oraz z pokoju 1 do pokoju 3 w kierunku strzałek.

Przetestowano również zasięg w linii prostej w korytarzu. Moduł HC-05 według noty katalogowej posiada zasięg około 10 metrów. Po zmierzeniu otrzymano zasięg około 25m. Jest to bardzo dobry wynik i zwiększa potencjał projektu.

Kolejny test to wysyłanie wiadomości w różnych konfiguracjach wraz z usuwaniem liter i zmianą znaków. W takim teście należało sprawdzić poprawność działania obu urządzeń, czy wiadomości dochodzą bez błędów. W trakcie testu poprawiono kilka drobnych usterek w kodzie, korzystając z trybu debugowania.

6. Podsumowanie

Celem pracy było zbudowanie bezprzewodowego, samowystarczalnego urządzenia z systemem operacyjnym napisanym własnoręcznie. Miało ono być zasilane bateryjnie oraz pozwalać na wysyłanie wiadomości tekstowych drogą radiową do drugiego lustrzanego urządzenia. Założenia projektu zostały spełnione, a ważniejsze funkcje urządzenia zostały szczegółowo opisane w pracy.

Praca polegała na złożeniu na płytce montażowej (później na płytce PCB) kompletnego systemu modułów komunikujących się ze sobą. Kolejnym etapem było programowanie i ten właśnie etap wymagał najwięcej czasu. Głównym założeniem dla tworzonego oprogramowania była jego prostota i niezawodna praca. Pisząc program okazywało się, że brakuje zabezpieczeń przykładowo przy wybieraniu większej liczby znaków lub przy odbieraniu wiadomości wtedy, gdy jednocześnie pisze się wiadomość. Pojawiały się problemy typu: przeskakujący kursor, litery wpisywane w niewłaściwe miejsce, wysyłanie artefaktów znakowych i inne tym podobne. Zaimplementowano, więc tryb debugowania, który nie był dopracowany pod kątem komunikatywności i stylu, ale okazał się bardzo pomocny podczas naprawiania błędów. W trakcie programowania pojawiały się kolejne pomysły na usprawnienie pracy systemu, takie jak szybkie pisanie czy wiadomości informacyjne.

W chwili skończenia systemu operacyjnego, pojawiła się przeszkoda w postaci wykonania płytki PCB. Nie był to pierwszy projekt elektroniczny autora, ale nigdy do tej pory nie był on wykonywany fizycznie. Po zapoznaniu się z techniką wykonywania druku proces wytrawiania dwuwarstwowej płytki (schemat można znaleźć w dodatku B) odbył się bez dalszych przeszkód. Pierwsza wersja płytki działała nie w pełni, gdyż wynikało to z błędów w projekcie. Kolejne dwie wersje już tych błędów nie posiadały.

Praca nad projektem trwała pół roku. Dzięki systematyczności termin jej oddania nie był przeszkodą ani bodźcem do przyśpieszenia pracy. Pracując przy projekcie zdobyto wiedzę na temat działania protokołu Bluetooth, tworzenia mini systemu operacyjnego oraz obsługi użytkownika i interakcji systemu z operatorem. Ważną zaletą jest pozyskanie umiejętności produkcja płytek PCB we własnym warsztacie. Dla elektronika jest to istotne.

Zalety opracowanego urządzenia to łatwość obsługi, mobilność oraz dobry zasięg. Wiadomości można pisać w obrębie domu rodzinnego oraz przekazywać z domu na zewnątrz. Urządzenie wydaje sygnały dźwiękowe. Pozwala to „usłyszeć” nadchodzącą wiadomość z pewnej odległości od urządzenia. Sieć jest bezpieczna dzięki hasłu oraz brakowi możliwości podłączenia do sieci jakiegokolwiek innego urządzenia poza tym z listy sparowanych.

Możliwości ulepszenia komunikatorów to zmiana wyświetlaczy na TFT, które pozwalają na wyświetlenie znacznie dłuższego tekstu lub po prostu dodanie przewijania tekstu. Można także zmienić moduł radiowy dysponujący większą mocą lub działający na innym paśmie radiowym. W zależności od wyboru wpłynie to istotnie na zasięg.

Zastosowania dla takiego urządzenia to komunikacja lokalna, w szczególności w chwili utraty prądu lub gdy sieć komórkowa bądź Internet przestaną funkcjonować. Istotne byłoby wtedy zwiększenie zasięgu do wymaganych potrzeb oraz, być może, użycie anten zewnętrznych na dane pasmo pracy układu.

Bibliografia

Wykaz w kolejności występowania w tekście odwołania:

[1] Arduino, https://www.arduino.cc/en/Guide/Introduction

[2] Wikipedia, *Architektura harvardzka*, https://en.wikipedia.org/wiki/Harvard_architecture

[3] Architektura Arduino http://www.edgefxkits.com/blog/wp-content/uploads/Arduino-Architecture.jpg

[4] Vasanth Vidyakar, *Understanding Arduino Architecture*, http://www.skyfilabs.com/blog/understanding-arduino-architecture

[5] *Schemat Arduino Nano* https://www.arduino.cc/en/uploads/Main/ArduinoNano30Schematic.pdf

[6] Nota katalogowa Atmega328p, rozdziały: 13, 17, 18, 19, 24

[7] AVR Guide *External interrupts on the Atmega328* https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328

[8] *Arduino Nano* http://www.etechpk.net/wp-content/uploads/2016/02/ARDUINO_NANO_03.png

[9] *Schemat pinów Arduino Nano* http://www.pighixxx.com/test/wp-content/uploads/2014/11/nano.png

[10] J. Szóstka, *Fale i anteny*, WKŁ Warszawa 2000, 2001

[11] Wikipedia, *Very High Frequency* https://en.wikipedia.org/wiki/Very_high_frequency

[12] J. Szóstka, *Mikrofale*, *1. Cechy charakterystyczne i wykorzystanie mikrofal*, WKŁ Warszawa 2006

[13] UKE, *Wykaz systemów radiowych*, https://www.uke.gov.pl/files/?id_plik=12366

[14] J. Szóstka, *Mikrofale*, *Dodatek A.*, WKŁ Warszawa 2006

[15] J. Szóstka, *Mikrofale*, *10.2 Anteny w łączach mikrofalowych.*, WKŁ Warszawa 2006

[16] Antena paraboliczna http://www.telesystem-world.com/uploads/imgs/List_item/12-17-2008-te80-2262.jpg

[17] Antena tubowa http://www.tdm-electronics.com/katalog3/d/22499-3/PICT9259.JPG

[18] Antena mikropaskowa https://en.wikipedia.org/wiki/Patch_antenna#/media/File:Antenne_patch_2.4_GHz.JPG

[19] J. Szóstka, *Mikrofale*, *12.4 System Bluetooth.*, WKŁ Warszawa 2006

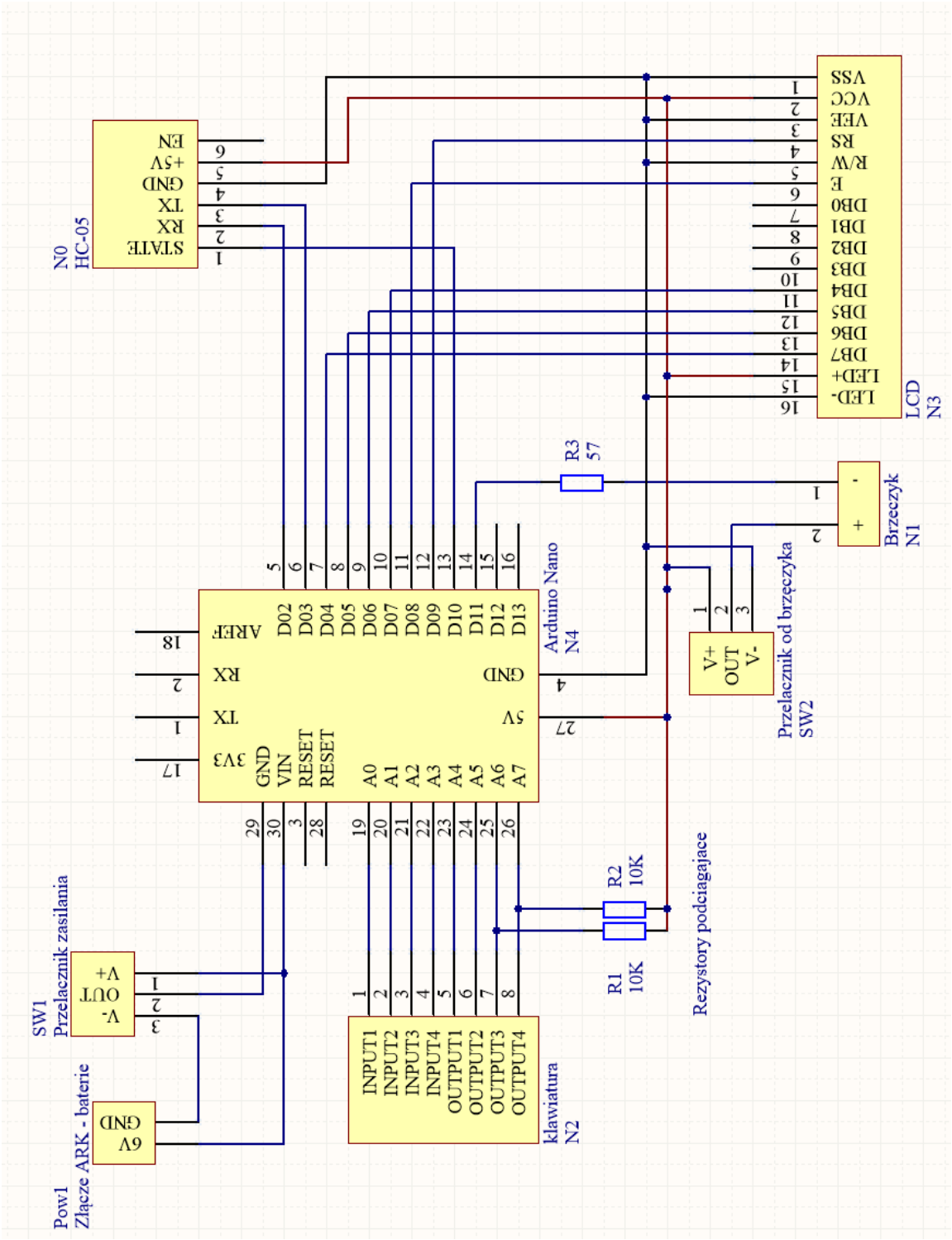
[20] HC-05 http://artofcircuits.com/wp-content/uploads/2016/07/HC-05-Bluetooth-Serial-MasterSlave.jpg

[21] Klawiatura membranowa, matrycowa, BOTLAND https://botland.com.pl/5411-thickbox_default/klawiatura-membranowa-numeryczna-16-klawiszy.jpg

[22] *SPECIFICATION OF LCD MODULE* http://botland.com.pl/index.php?controller=attachment&id_attachment=183

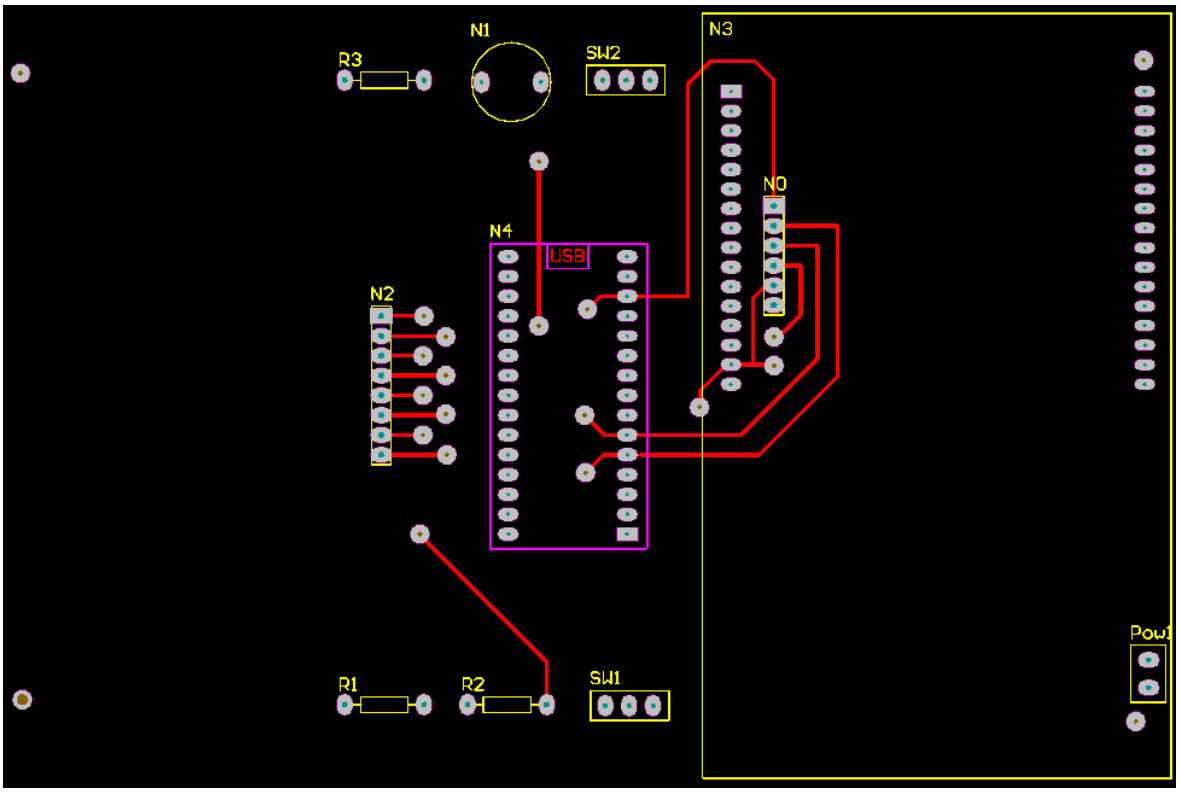
[23] LCD pinout https://www.robomart.com/image/catalog/RM0122/16x2-character-lcd-display-for-arduino-raspberry-pi-robotics-robot-display-devices-rm0122-by-robomart-a17713.jpg

Dodatek A. Schemat ideowy

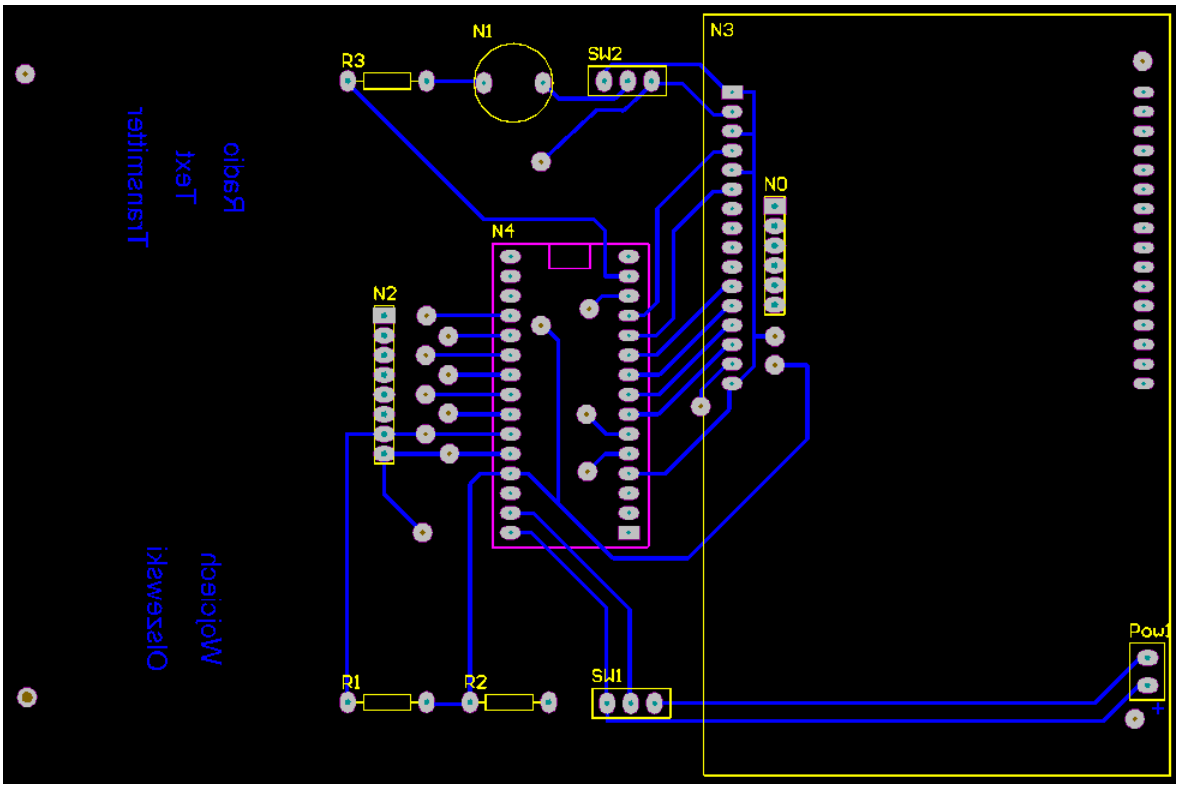


Dodatek B. Projekt PCB

Top Layer

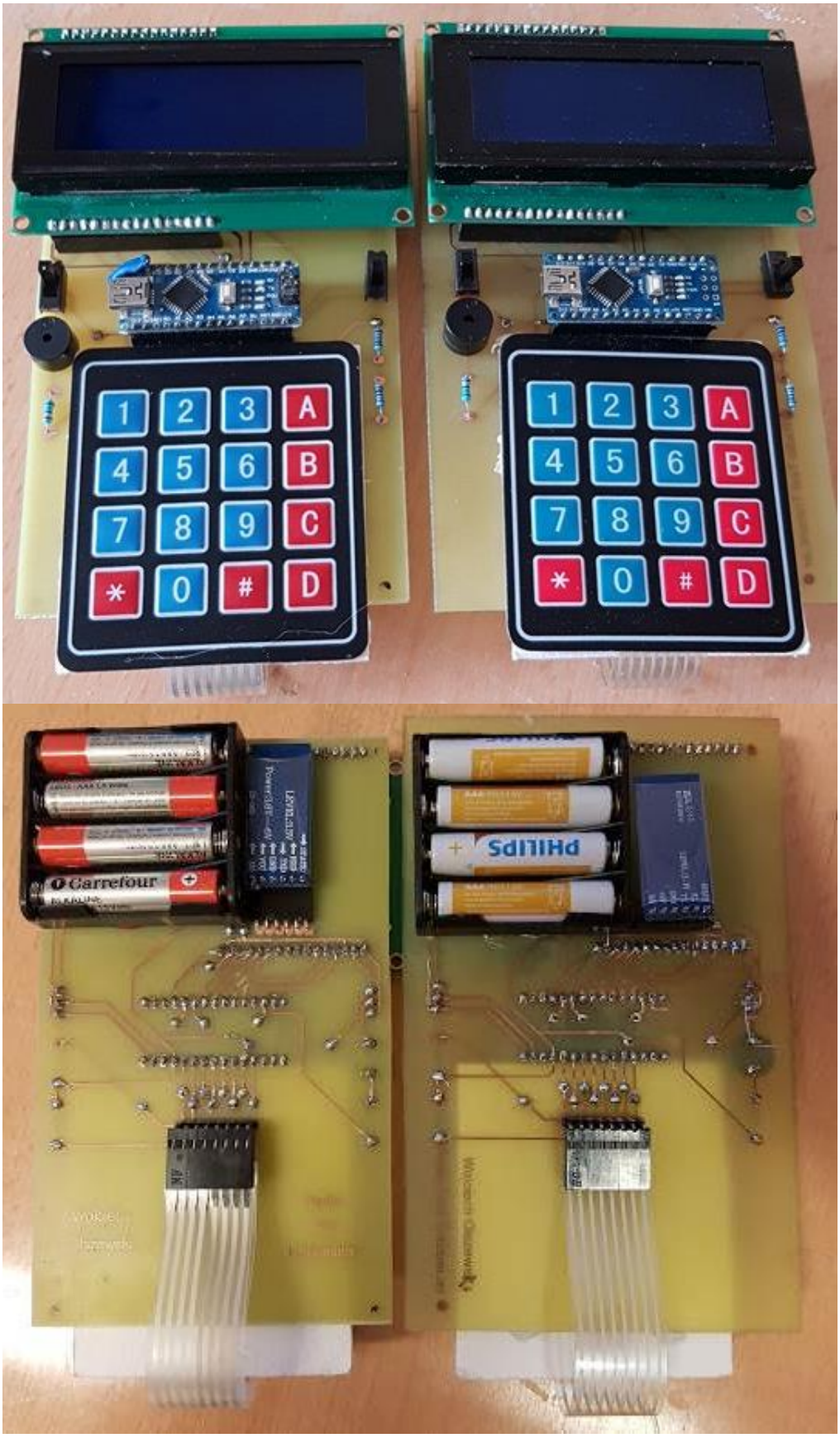


Bottom Layer



Dodatek C. Schemat znaków klawiatury

Dodatek D. Zdjęcia urządzeń



Dodatek E. Wykaz elementów nadajnika/odbiornika

- Platforma Arduino Nano
- Wyświetlacz LCD 4x20
- Klawiatura membranowa
- Moduł Bluetooth HC-05
- Brzęczyk
- 2 x przełącznik 2 stanowy
- 2 x rezystor 10K
- 1 x rezystor
- Koszyk na baterie AAA
- 4 x bateria AAA
- 2 x złącze goldpin męskie
- 4 x złącze goldpin żeńskie
- Złącze goldpin męskie kątowe
- Złącze goldpin żeńskie kątowe
- Złącze ARK podwójne raster 5mm
- Laminat dwuwarstwowy, wymiary około 10x15cm