Spis treści

[1. Cel pracy dyplomowej 1](#_Toc115045185)

[2. Rejestrator rowerowy – źródło danych 1](#_Toc115045186)

[a) Budowa urządzenia 1](#_Toc115045187)

[b) Zasada działania 3](#_Toc115045188)

[c) Ramka danych 5](#_Toc115045189)

[3. Aplikacja – analizator aktywności rowerowej 6](#_Toc115045190)

[a) Ogólne dane 6](#_Toc115045191)

[b) Wygląd i interface użytkownika 6](#_Toc115045192)

[c) Komunikacja – pozyskiwanie danych 7](#_Toc115045193)

[d) Wizualizacja danych na wykresach 12](#_Toc115045194)

[e) Wymagania sprzętowe 14](#_Toc115045195)

[f) Zalety używania aplikacji 14](#_Toc115045196)

# Cel pracy dyplomowej

Praca dyplomowa została podzielona na dwie części: cel właściwy oraz cel poboczny. Celem właściwym było stworzenie aplikacji okienkowej napisanej w języku C#, która ma za zadanie przeanalizować dane dostarczone przez cel poboczny, którym jest urządzenie (zwane danej rejestratorem rowerowym) zbudowane na 8-bitowym mikrokontrolerze i oprogramowane w języku Bascom.

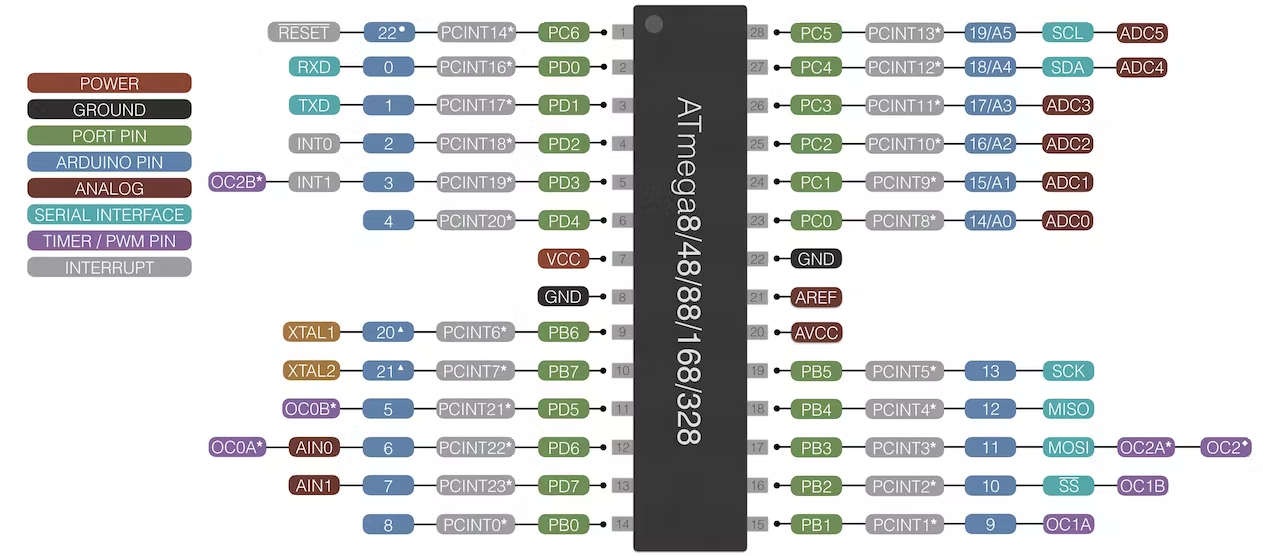
# Rejestrator rowerowy – źródło danych

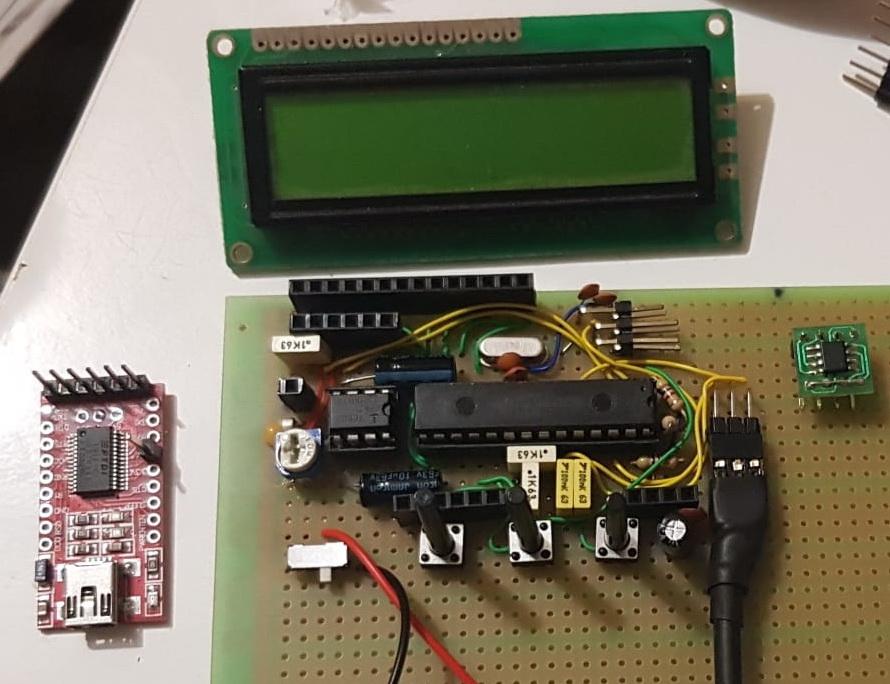
## Budowa urządzenia

Rolę rejestratora danych pełni urządzenie stworzone od zera i indywidualnie oprogramowane. „Sercem” układu jest 8-bitowy mikrokontroler firmy Atmel – Atmega88-20. Poniżej zaprezentowane zostały podstawowe parametry tego chipu:

|  |  |
| --- | --- |
| Producent | **MICROCHIP (ATMEL)** |
| Typ układu scalonego | **mikrokontroler AVR** |
| Rodzina | **ATMEGA** |
| Pojemność pamięci EEPROM | **512B** |
| Pojemność pamięci SRAM | **1kB** |
| Pojemność pamięci Flash | **8kB** |
| Obudowa | **PDIP-28** |
| Częstotliwość taktowania | **20MHz** |
| Liczba wejść/wyjść | **23** |
| Liczba kanałów PWM | **6** |
| Liczba timerów 8bit | **2** |
| Liczba timerów 16bit | **1** |
| Montaż | **THT** |
| Temperatura pracy | **-40...85°C** |
| Częstotliwość pracy max | **20MHz** |
| Liczba przerwań zewnętrznych | **24** |
| Liczba kanałów output compare | **6** |
| Liczba komparatorów | **1** |
| Liczba kanałów input capture | **1** |
| Liczba przetworników A/D 10bit | **8** |

Zdjęcie wyjść układu Atmega88:



Zdjęcie z późniejszej budowy urządzenia:

**6**

**9**

**10 2 11 3**

**4 4 1**

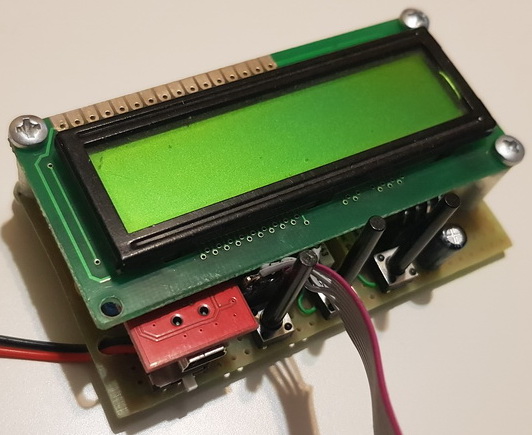
**12 13 14**

**5 8**

**7**

Na zdjęciu widzimy:

1. Układ główny Atmega88
2. Zegar układu – rezonator kwarcowy 16MHz
3. Pamięć FRAM o pojemności 256kbit czyli 32768B gotów pomieścić maksymalnie 4,5h rejestracji
4. Konwerter napięcia dla wyświetlacza (z powodu konfliktu napięć – układ 3.3V, wyświetlacz 5V)
5. Konwerter interfejsu USB<>RS232 (gotowy moduł) do komunikacji z komputerem i ładowania baterii
6. Wyświetlacz alfanumeryczny 2x16 znaków
7. Przyciski – intefrace użytkownika (lewo, ok, prawo)
8. Włącznik główny
9. Złącze wyświetlacza
10. Złącze konwertera USB
11. Złącze czujników: hamulca lewego, hamulca prawego, obrotu koła
12. Złącze programatora układu
13. Złącze pamięci FRAM
14. Złącze czujników: obrotu korby oraz liniowych od przerzutek

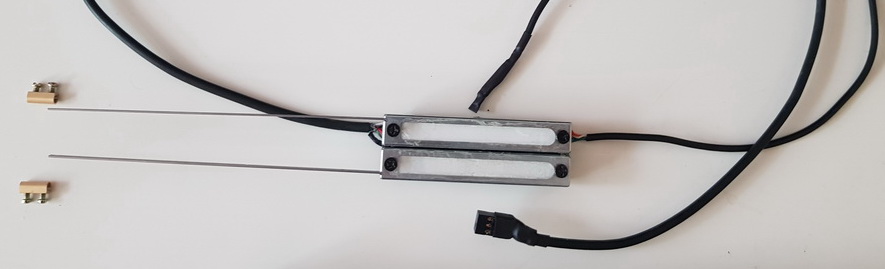
Złożone urządzenie: 

Rolę czujników w układzie pełnią 4 kontaktrony i dwa potencjometry liniowe. Są to szklane rurki z hermetycznie zamkniętymi w środku dwoma stykami stalowymi, które reagują na pole magnetyczne. Normalnie kontaktron jest rozwarty i nie przewodzi prądu. W momencie pojawienia się źródła pola magnetycznego (u nas magnes przyczepiony do koła, klamki rowerowej czy korby) płytki w elemencie zbliżają się do siebie, zwierają i w obwodzie płynie prąd, co z kolej jest rejestrowane przez układ. W ten sposób rejestrator jest informowany o przyciśnięciu hamulca, jednym obrocie koła czy korby. Troszkę inaczej odbywa się nadzór nad zmianą przełożenia w rowerze. Tu rolę czujników spełniają przerobione potencjometry liniowe, których suwaki zaczepione są o linki przerzutek. Odczyt aktualnego biegu odbywa się przez pomiar napięcia na potencjometrze. Przy pomocy wewnętrznego 10-bitowego przetwornika analogowo cyfrowego, układ Atmega88 rejestruje wartość od 0 do 1023, następnie dana jest przeliczana na zakres 1-3 dla biegów przednich bądź 1-9 dla biegów tylnych. Przełożenie 2x5 będzie zatem zapisane jako łączna liczba 25 (6-bitów). Po zgraniu danych do aplikacji będzie ona miała za zadanie odszukanie w bajcie danych właściwych bitów i przekonwertowanie ich wartości na zapis 2x5.

Zdjęcie przetworników biegów i jednego czujnika kontaktronowego w osłonie:

1

2

1. Kontaktron rurkowy w obudowie 2. Dwa przerobione potencjometry liniowe

## Zasada działania

Głównym zadaniem urządzenia jest rejestracja parametrów w trakcie jazdy na rowerze. Co sekundę zapisywane jest każde naciśnięcie hamulca lewego czy prawego, czas jednego obrotu koła, ilość obrotów korby oraz aktualne przełożenie. Jednocześnie wszystkie parametry prócz obrotu korby możemy obserwować na wyświetlaczu. Urządzenie ma też funkcję samego licznika bez rejestrowania danych. Poniżej prezentacja przejścia przez całe menu:

 ►

▼

►

▼



►

▼

►

▼

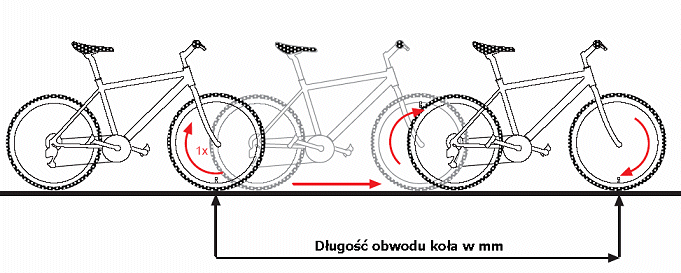
►

▼

►

**Prędkość**

Głównym parametrem rejestrowanym przez urządzenie jest prędkość. Aby ją wyliczyć potrzebujemy dwóch parametrów: drogi i czasu, w którym ta droga została pokonana. W naszym układzie dysponujemy tymi danymi – posiadamy informację o pełnym obrocie koła jak również czas zmierzony przez 16-bitowy timer. Wiemy również, że w układzie znajduje się rezonator kwarcowy, który taktuje układ z częstotliwością 16MHz. To znaczy, że timer będzie zliczał czas z interwałem 1/16.000.000 sekundy. Jeśli chcielibyśmy odliczyć 1 sekundę, to musielibyśmy sprawdzać timer, aż do momentu gdy jego wartość będzie równa 16.000.000. Jest to wartość za duża aby 16-bitowy timer mógł ją zliczyć, dlatego też programiści pomyśleli o preskalerach czyli dzielnikach, które podzielą główny sygnał zegarowy i pozwoli na zmierzenie dłuższych czasów. Ja użyłem maksymalnego dzielnika czyli 1024. Dzięki temu jedna sekunda to 16.000.000/1024=15625 taktów. Zyskujemy zatem rozdzielczość 0,000064 sekundy pomiaru. Użytkownik sam podaje zmierzony obwód koła np. 2077mm. Jadąc na rowerze dostajemy informację o minięciu czujnika przez magnes zainstalowany na jednej ze szprych koła. W tym momencie zapisywany jest stan timera w zmiennej, sam timer jest zerowany, a dana leci dalej do przeliczenia. Dzielimy zatem obwód koła przez ilość czasu zmierzonego, przeliczamy na km/h, zapisujemy i pokazujemy na wyświetlaczu.



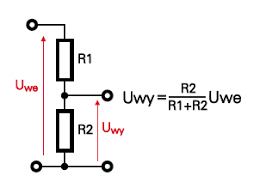
**Kadencja**

Kolejnym monitorowanym parametrem jest kadencja, czyli ilość obrotów korbą w ciągu ostatnich 60 sekund. Odbywa się to jednak w sposób inny niż pomiar prędkości za pomocą timera. Tu po prostu mierzymy ilość obrotów korby w ciągu sekundy. W rzeczywistości parametr kadencji u kolarzy nie przekracza 120obr/min czyli 2obr/s.

**Hamulce**

Pomiar hamowania tak jak poprzednie odbywają się za pomocą kontaktronów i mają charakterystykę ON/OFF. Nie zakładam (chociaż jest to możliwe), że hamowanie będzie odbywało się w sposób pulsacyjny częstszy niż 1Hz. Dlatego też tu pomiar odbywa się na zasadzie „pierwszego strzału”. Czujniki podłączone są do wyjścia zewnętrznego przerwania w mikrokontrolerze. W momencie naciśnięcia klamki, układ ustawia wartość zmiennej bool-owskiej na *true* i tak już zostaje do końca trwania 1 sekundy, bez względu na ilość kolejnych hamowań. Po tym czasie wszystkie flagi są zerowane i proces może zacząć się od nowa.

**Przełożenia**

Odczyt przełożeń odbywa się w inny sposób. Tu problemem było zagadnienie jak odczytać (w mniej korzystnym przypadku) 9 położeń, gdyż tyle jest zębatek w tylnej kasecie. 9 stanów to zakres od 0 do 8, a więc gdyby chciało się zastosować jakiś przetwornik to zajęło by to minimum 4 linie danych (8 to binarne 1000). Można by było zastosować przetwornik z wyjściem szeregowym. To natomiast rozwiązanie, które potrzebuje minimum 2 linie – zegarową i danych. Biorąc pod uwagę, że potrzebujemy dwie sztuki, musimy podwoić ilość linii. Najtańszym i najlepszym rozwiązaniem było skorzystanie z potencjometrów w połączeniu z inwentarzem mikrokontrolera – przetwornikiem analogowo-cyfrowym. W ten sposób można zrealizować teoretycznie tyle stanów na jednej linii ilu bitowy jest przetwornik. My dysponujemy 10-bitowym, jednak ja konfiguruję go na wersję minimalną 8-bitową, bo taka większa dokładność nie jest nam potrzebna. Do wyjść potencjometrów podłączone jest zasilanie, a środkowy (ruchomy) odczep połączony jest do pinu przetwornika ADC. Potencjometr w takim układzie to swoisty dzielnik napięcia – napięcie na odczepie jest proporcjonalne do jego odległości od końcówki 0V. Zależność ukazuje rysunek z boku. Tym sposobem dwoma liniami mierzymy wszystkie przełożenia dwóch przerzutek.

**Pomiar czasu**

Odmierzenie jednej sekundy nie wymaga w naszym układzie zewnętrznych komponentów. Chip posiada dwa timery. Jeden został wykorzystany przy pomiarze prędkości, więc tu wykorzystamy drugi. Tutaj w odróżnieniu od pomiarów przy prędkości nie zliczamy impulsów od zera tylko ustawiamy wartość początkową licznika i czekamy na jego przepełnienie. Przepełnienie wywołuje przerwanie, a w przerwaniu mamy funkcję zbierającą wszystkie parametry i zapisującą do pamięci zewnętrznej FRAM. Jaka będzie wartość początkowa? To trzeba wyliczyć samemu. Tak jak poprzednio – licznik może zliczyć liczbę 16-bitową czyli maksymalna wartość 65535, używamy preskalera 1024 czyli timer mamy taktowany częstotliwością 16.000.000/1024=15625Hz. Zatem jedna sekunda to 15625 taktów zegara. Aby przepełnić licznik po czasie 1s i wywołać przerwanie, musimy ustawić wartość początkową równą 65535-15625=49910. Cały cykl się powtarza:

ustawienie wart. początkowej → zapis pomiarów → przepełnienie → ustawienie wart. początkowej → …

## Ramka danych

Jednym z zagadnień, z którym trzeba się zmierzyć w dzisiejszym świecie jest temat „upchania” jak największej ilości informacji w jak najmniejszej ilości bajtów. Również tutaj występuje ten problem. Mamy do zapisania następujące dane:

* prędkość - wartości z przedziału 0 – ~70.0 (z częścią dziesiętną)
* hamulec lewy i prawy – wartość ON/OFF
* ilość obrotu korby w ciągu 1s – wartość z przedziału 0 – 2
* stan przełożeń – wartości z przedziału 11 – 39

Nasza pamięć FRAM jest kością 8-bitową, która pomieści 32768 bajtów danych. Im bardziej je „spakujemy tym dłuższą wycieczkę możemy zarejestrować.

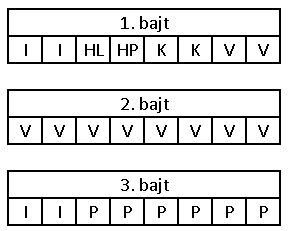
Czytając wiele artykułów podjąłem decyzję, że prędkość najlepiej będzie zapisać nie w postaci liczby zmiennoprzecinkowej, która zajmie 4 bajty, a po prostu przemnożyć ją przez 10 i zapisać jako liczbę całkowitą. Czyli wartość 35,2 zapisana będzie w formacie 352 co przy zakresie 0 – 70 będzie wartościami 0 – 700, a to zajmie bitowe 10 miejsc. Biorąc to pod uwagę, teoretycznie, możemy maksymalnie zapisać wartość 102.4km/h. Teoretycznie, gdyż występują jeszcze inne ograniczenia układu np. czas na zapis do pamięci, czas na pomiary itp.

Hamulce są z przedziału wartości 0 – 1, a więc tu nie ma problemu, po prostu każdy zajmie po jednym bicie.

Ilość obrotów korby nie zmieści się w jednym bicie – potrzeba ich, aż dwóch, aby zapisać cyfrowo liczbę 2.

Doszliśmy do stanu przełożeń. Tu jest pytanie dodatkowe – czy potrzebujemy rejestrować w każdej sekundzie stan przełożenia? Przecież wystarczy to zrobić tylko wtedy, gdy nastąpi zmiana. Tak też ustaliłem, jednak to ma też swoje konsekwencje. Ponieważ ilość bajtów w danej sekundzie będzie różna, trzeba również ustanowić system w jakim bajty będą rozpoznawane. Czasami będzie ich więcej, a czasami mniej – trzeba wiedzieć który Bajt co reprezentuje, żeby np. nie odczytać bajtu reprezentującego dane przełożeń jako bajtu prędkości. Tym sposobem dochodzimy do wniosku, że potrzebujemy prefixu. Prefix czyli przedrostek, po którym będziemy rozpoznawali dany bajt. Ustaliłem, że przeznaczę dwa początkowe bity na tę informację. Wracając do kwestii zakresu wartości przełożeń liczba 39 zajmie nam 6 bitów. Wartość tą moglibyśmy jeszcze „ścisnąć” uwzględniając zero, czyli dla maksymalnego przełożenia 3x9 ustalić nie wartość 39 a 28. Wtedy zakres przełożeń przednich byłby równy 0 – 2 a tylnych 0 – 8. Pytanie czy będzie nam to potrzebne.

Po długiej analizie ustaliłem, że pełna ramka danych, rejestrowana co sekundę, będzie miała 3 bajty, a niepełna (nie zawierająca aktualnego przełożenia) 2 bajty. Wizualnie wygląda to następująco:

I – informacja o bajcie (prefix)

HL – bit hamulca lewego

HP – bit hamulca prawego

K – bity korby

V – bity prędkości

P – bity przełożenia

Zasada jest taka, że bajty 1 i 2 występują zawsze, a bajt 3 tylko wtedy, gdy zauważymy zmianę i do jego rozpoznania potrzebny nam właśnie prefix. Prefix dla bajtu 1 to 00, a dla przełożeń 01.

# Aplikacja – analizator aktywności rowerowej

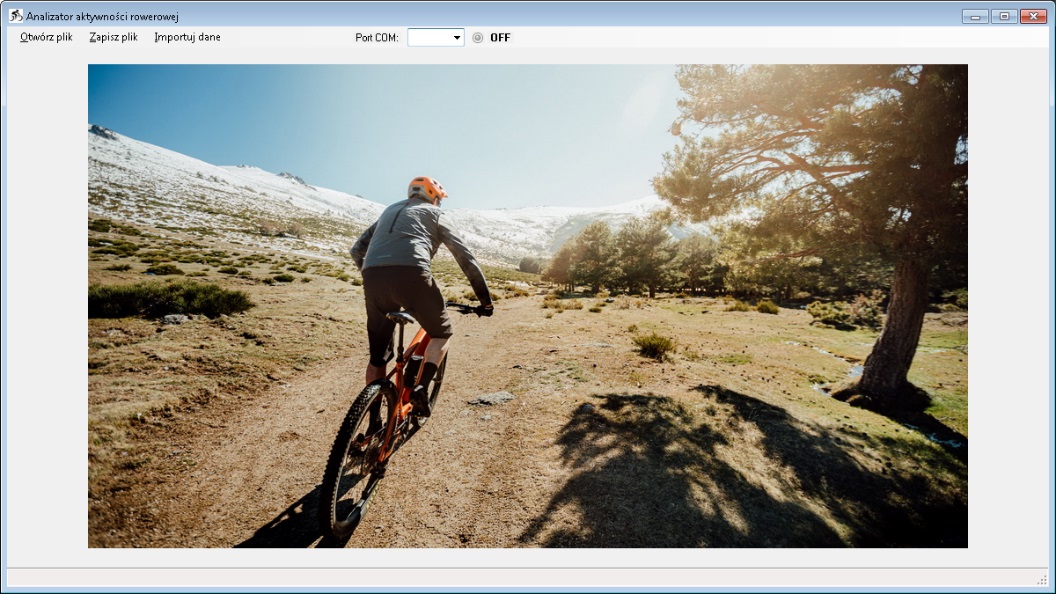
## Ogólne dane

Aplikacja napisana jest w języku C# w programie Visual Studio. Analizator aktywności rowerowej jak sama nazwa wskazuje ma za zadanie przeanalizować dostarczone dane, wyszczególnić podstawowe parametry całej wycieczki i zwizualizować je. Aplikacja jest w pełni responsywna, czyli skalowalna. Podstawowe funkcje jakie posiada program to:

* możliwość zaimportowania danych z rejestratora
* możliwość zapisania danych na dysku
* możliwość wczytania danych z dysku
* analiza wczytanych danych

## Wygląd i interfejs użytkownika

Ekran powitalny

 1 2 3 4

Menu główne programu:

1. Otwiera pliki programu z rozszerzeniem .aar [2]
2. Zapisuje dane do pliku z rozszerzeniem .aar [2]
3. Wczytuje dane z rejestratora [1]
4. Wybór portu komunikacyjnego COM [1]

Ekran właściwy programu



2

1

3

4

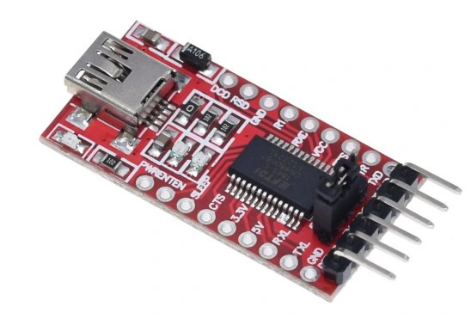
1. Główny panel wykresów
2. Ogólne parametry wycieczki, wartości maksymalne i średnie
3. Diagramy użycia poszczególnych przełożeń
4. Pasek wartości chwilowych odczytywanych w miejscu zaznaczonym pionową czerwoną linią

## Komunikacja – pozyskiwanie danych

O ile funkcja otwierania i zapisu do pliku nie wymaga wyjaśniania o tyle sam system komunikacji i zgrywania danych z urządzenia jest już ciekawy. Ogólnie interfejsem wykorzystanym do komunikacji jest uniwersalna magistrala szeregowa, czyli popularne USB. Jest to bardzo szybki interfejs, którego transfery dochodzą aktualnie do 40Gb/s (USB 3 Gen 3x2). [3] Nie ma chyba już teraz urządzeń, które nie posiadają tego złącza i to było moim głównym powodem wyboru. Złącze to jest również wykorzystywane do ładowania akumulatora urządzenia. Rejestrator nie potrzebuje aż tak wysokich transferów, ponieważ maksymalna wielkość przesyłanych danych, przy pełnym zapełnieniu pamięci to 3x 32kB. Kolejną kwestią jest wyposażenie mikrokontrolera. Nie posiada on sprzętowego wejścia USB. Do wyboru były następujące interfejsy:

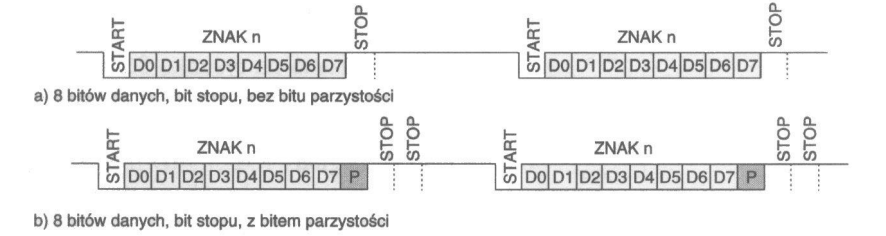
* SPI (Serial Peripheral Interface) – szeregowy interfejs urządzeń peryferyjnych. Jeden z najczęściej używanych interfejsów komunikacyjnych pomiędzy systemami mikroprocesorowymi, a układami peryferyjnymi takimi jak: przetworniki ADC/DAC, układy RTC, pamięci EEPROM, pamięci flash, karty MMC/SD/ itp. Do komunikacji wykorzystuje on 3 linie danych. [4]
* I²C (Inter-Integrated Circuit) – szeregowa, dwukierunkowa magistrala służąca do przesyłania danych w urządzeniach elektronicznych. Została opracowana przez przedsiębiorstwo Philips na początku lat 80. Standard I²C określa dwie najniższe warstwy modelu odniesienia OSI: warstwę fizyczną i warstwę łącza danych. Do komunikacji wykorzystuje on 2 linie danych.[5]
* RS232 (Recommended Standard) – magistrala komunikacyjna przeznaczona do szeregowej transmisji danych. Opracowana w 1962 roku na zlecenie amerykańskiego stowarzyszenia producentów urządzeń elektronicznych (Electronic Industries Alliance) w celu ujednolicenia para- metrów sygnałów i konstrukcji urządzeń zdolnych do wymiany danych cyfrowych za pomocą sieci telefonicznej. Najbardziej popularna wersja tego standardu, RS-232C pozwala na transfer na odle- głość nie przekraczającą 15m z szybkością maksymalną 20 kb/s. Wykorzystuje 2 linie danych. [6]

Ostateczny wybór padł na interfejs RS232. Chcąc pogodzić możliwość komunikacji między układem, a komputerem konieczne było zakupienie popularnej przejściówki RS232<>USB. Na płytce znajduje się wejście usb mini (niestety mini a nie C), specjalizowany układ firmy FTDI - FT232RL, diody sygnalizujące transfer danych i zasilanie oraz zworę ustawiającą napięcie zasilania. Rozwiązanie to ma parę zalet, a mianowicie jest małe oraz ma możliwość konwersji napięć usb 5V<>3.3V rejestrator.

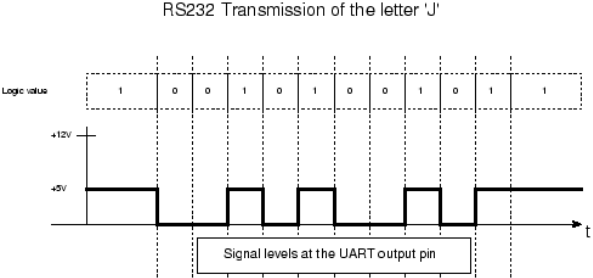


Na stronie producenta układu znajduje się zalecany sterownik, który instalujemy na komputerze. Sterownik po wykryciu podłączenia układu rozpoznaje go i tworzy w systemie wirtualny port COM. Od tej chwili warstwę fizyczną mamy już przygotowaną. Kolejnym etapem jest oprogramowanie urządzenia i zaimplementowanie w aplikacji możliwości komunikacji szeregowej.

Wracając do samego standardu komunikacji, który został zaimplementowany w aplikacji trzeba wspomnieć, iż RS 232C stosowany bywa wszędzie tam, gdzie mniej istotną rolę odgrywa przepustowość łącza, natomiast ważna jest niezawodność i prostota obsługi protokołu komunikacyjnego. Komputery osobiste wyposażone są w łącza szeregowe przystosowane do transmisji asynchronicznej, tzn. komputer i urządzenie muszą pracować z jednakową, wcześniej uzgodnioną prędkością oraz taką samą strukturą znaków. Transmisja taka może być realizowana w trybie bez potwierdzenia odbioru lub z potwierdzeniem odbioru. Drugi sposób zapewnia nam możliwość kontrolowania poprawności wysyłanych–odbieranych danych. Dane przesyłane są w postaci tzw. ramki (ang. frame), która jest najmniejszą porcją możliwej do przesłania informacji. Bity przesyłane są kolejno. Do kodowania znaków stosuje się najczęściej kod ASCII (American Standard Code of Information Interchange).[7]Obecnie ramka może zawierać od 5 do 8 bitów danych (jednak większość spotykanych urządzeń posługuje się słowem 7 lub 8 bitowym) poprzedzonych bitem startu oraz zakończonych bitem parzystości i jednym lub więcej bitami stopu. Przed rozpoczęciem transmisji bit startu przyjmuje zawsze wartość 0, zaznaczając wyraźnie moment początkowy. Odwrotność czasu trwania transmisji jednego bitu określa szybkość przesyłu w bitach na sekundę.



Bit kontroli parzystości przesyłany za ostatnim bitem danych jest jedną z metod monitorowania poprawności transmitowanych danych. Z reguły przyjmuje dwie wartości: 0 lub 1. Ilości jedynek w polu danych może być uzupełniana do liczby parzystej (evenparity) lub nieparzystej (oddparity). Łącze w trakcie ciszy utrzymywane jest w stanie logicznej 1. Transmisja rozpoczyna się od bitu startu, który zawsze przyjmuje wartość logicznego 0. Po nim następuje transmisja ośmiu bitów reprezentujących znak. Później jest bit parzystości, potem dwa bity stopu zamykające ramkę. Po bicie stopu łącze wraca do stanu ciszy. Jeden lub dwa bity stopu stosowane są po to, by odbiornik i nadajnik mogły dokonać wzajemnej synchronizacji przed transmisją kolejnej ramki danych.

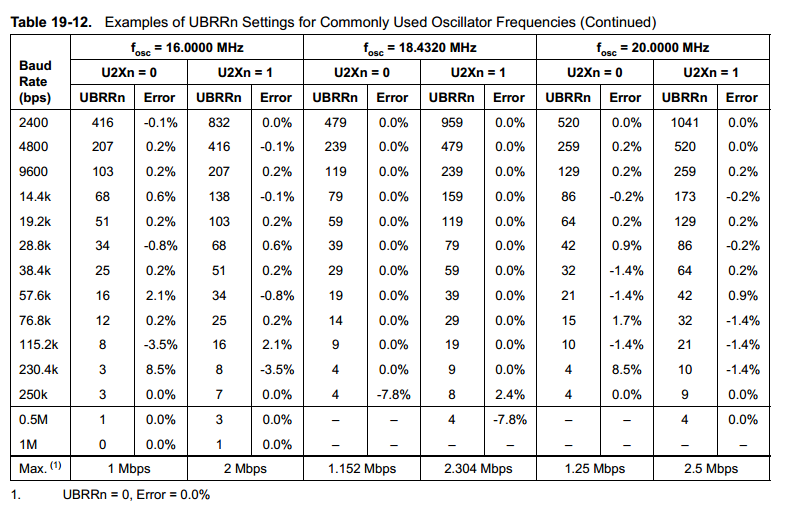


W standardzie IBM wykorzystuje się jedynie 9 sygnałów, które są wystarczające do zrealizowania transmisji asynchronicznej. Linie TxD oraz RxD są przeznaczone do obustronnego przesyłania danych. Nazywamy je liniami danych. Pozostałe zaś są liniami sterującymi lub kontrolnymi (oczywiście za wyjątkiem linii masy). Ogólnie sygnały przekazywane łączem RS 232C można podzielić na trzy grupy:

* sygnały danych: RxD, TxD,
* sygnały sterujące urządzeniem zewnętrznym: RTS, DTR,
* sygnały odbierane od urządzenia (kontrolne): CTS, DSR, RI, RLSD (DCD).

Do podstawowej komunikacji wystarczą jednak jedynie linie TxD oraz RxD, które podczas komunikowania się dwóch układów (np. mikrokontrolerów) powinny być skrzyżowane: wyjście TxD pierwszego układu powinno być dołączone do wejścia RxD drugiego układu, a wyjście TxD drugiego układu połączone z wejściem RxD pierwszego układu.

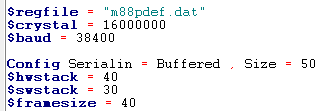
Podczas wybierania prędkości transmisji trzeba uwzględnić również częstotliwość taktowania układu. Nie wszystkie prędkości są korzystne i w jednych mogą występować większe a w innych mniejsze błędy w transmisji. Poniżej przedstawiam tabelę [8] kompatybilności wartości baud rate do częstotliwości zegara w układzie:



Przechodząc do implementacji komunikacji w naszej aplikacji prezentuję część odpowiedzialną za konfigurację przesyłu danych zawartą w funkcji otwierającej port oraz funkcję, która go zamyka. [1]

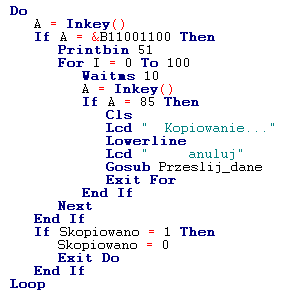
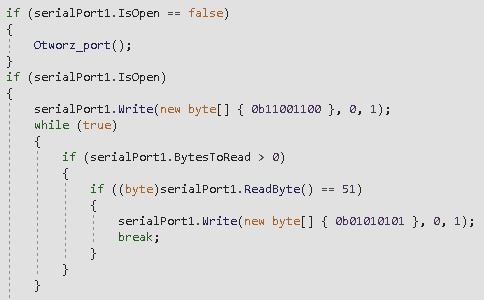


Po dodaniu w Visual Studio kontrolki portu szeregowego należy go skonfigurować. Jak widać u nas port został skonfigurowany na prędkość 38400b/s, 8 bitów danych, jeden bit stopu i brak parzystości. Tak jak napisałem wcześniej, oba źródła muszą być ustawione na te same parametry. Oto fragment kodu z programu Bascom AVR-a konfigurującego urządzenie:

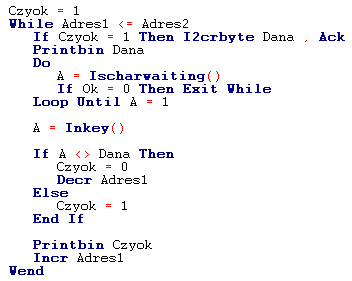
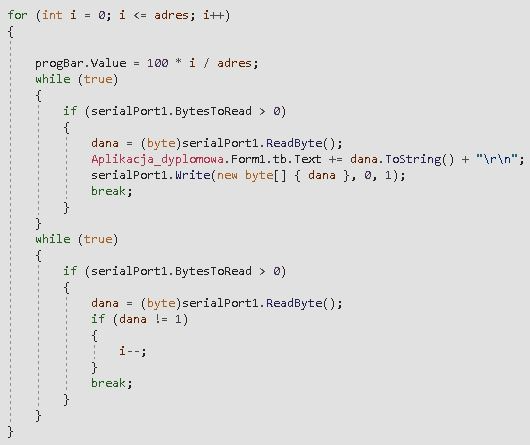


Cała próba otwarcia portu została zawarta w klauzuli TRY – CATCH na wypadek, gdyby port nie był dostępny. Jeśli program napotka na jakiś problem, poinformuje o tym użytkownika standardowym oknem błędu z informacją o konkretnej przyczynie jego wywołania. W takim przypadku przycisk radialny (rBtnON) który pełni funkcję kontrolki przestanie być aktywny, a status etykiety przy nim pozostanie z wpisem „OFF”. Kwestia zamknięcia portu nie potrzebuje takiej kontroli. Jeśli port nie jest otwarty, próba jego zamknięcia nie wywołuje błędów.

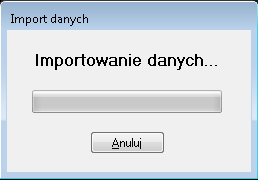
Żeby zaimportować dane z urządzenia należy wybrać z menu opcję „Zgraj dane” i zatwierdzić środkowym przyciskiem. Na wyświetlaczu pojawi się napis „Podłącz USB i uruchom program”. Od tej pory urządzenie jest w stanie nasłuchu. Po uruchomieniu aplikacji, należy najpierw wybrać port COM, a następnie w menu na belce kliknąć przycisk „Importuj dane” (odwrotna kolejność będzie skutkowała wyświetleniem stosownego komunikatu). Po prawidłowym zainicjowaniu otwarcia portu komunikacyjnego (właściwego dla naszego rejestratora) następuje sekwencja rozpoznawcza. Aplikacja wysyła do portu komunikacyjnego znak o numerze 204 (11001100b). Jeśli nasłuchujące urządzenie rozpozna ten znak, wysyła jego zanegowaną wersję, czyli 51 (00110011b), aplikacja odsyła wartość 85 (01010101b) i następuje zgrywanie danych. Tak wyglądają fragmenty kodu odpowiedzialne za inicjalizację kopiowania (odpowiednio z Bascom-a i Visual Studio):

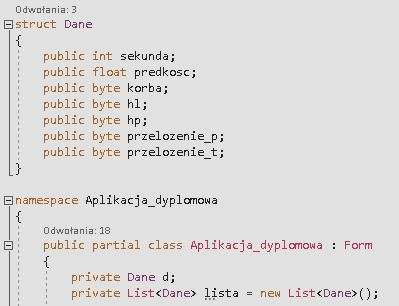
Zgrywane dane są kontrolowane. Każdy wysłany z urządzenia bajt jest do niego odsyłany, następnie urządzenie sprawdza czy to co wysłało i odebrało nie różni się od siebie. Jeśli nie to odsyła „1” i wysyła następny bajt, jeśli dane się różnią, odsyła „0” i powtarza wysyłkę jeszcze raz. Proces potraja dane, jednak mamy pewność, że wszystko zgrało się poprawnie. Oto fragmenty kodu z obu programów:



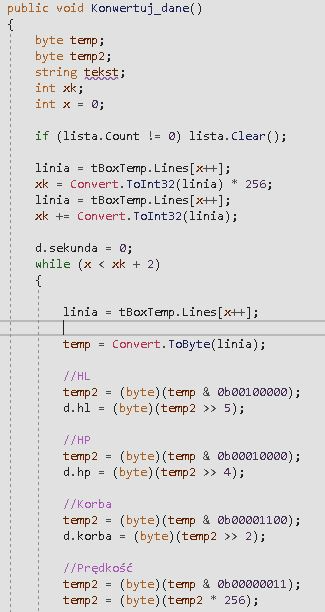
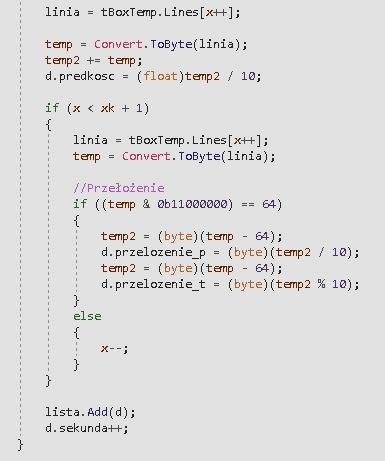
Ogólnie zgrywanie danych rozpoczyna się od nadania dwóch bajtów, które wartością reprezentują ilość danych jaka będzie podawana. Pozwoliło to na kontrolę przesyłania danych w postaci paska postępu jak na zdjęciu poniżej:



Wszystkie dane trafiają do textBox-a, skąd następnie są konwertowane i zapisywane do ostatecznej listy struktur. [14][15] Zdefiniowana struktura wygląda jak na obrazku poniżej:



Interesującym miejscem jest też funkcja konwersji danych. Konkretne bity są wyłuskiwane z bajtu za pomocą mnożenia bitowego – funkcji AND (&), która działa na zasadzie maski. Następnie bity są przesuwane do początku bajtu. Dobrze widać to na zdjęciu poniżej:

Ponieważ prędkość jako jedyna jest wartością zapisaną na 10 bitach, trzeba było ją „złożyć” w jedną zmienną typu integer. Następnym krokiem jest sprawdzenie prefixu kolejnego bajtu. Jeżeli prefixem jest 00 to mamy do czynienia z daną prędkości, jeśli 01 to mamy bajt przełożenia.

## Wizualizacja danych na wykresach

Z założenia głównym zadaniem aplikacji miała być wizualizacja danych liczbowych zebranych podczas jazdy. Po wczytaniu/zaimportowaniu danych oczom użytkownika ukazuje się pięć wykresów liniowych oraz dwa kołowe. Idąc od góry jest to prędkość, kadencja, przyśpieszenie, wykorzystanie hamulców prawego i lewego. Z prawej strony mamy dwa wykresy przedstawiające wykorzystanie poszczególnych biegów (tarcz) przerzutek przednich i tylnych. Wszystkie wykresy liniowe są responsywne (skalowalne) gdyż zostały umieszczone na panelu tablicowym (TableLayoutPanel) z narzuconymi proporcjami oraz właściwością dociągającą Dock = Fill (tak samo jak każdy wykres). [9]



Za wypełnienie danych w wykresach odpowiada funkcja Wypełnij\_wykresy(), jednak zanim to nastąpi część danych została przygotowana w osobnych tablicach. Lista ze strukturą Dane zawiera jedynie informacje wyłuskane z surowych bajtów danych, ale nie są one jeszcze obrobione. Właściwościami, które wymagały dodatkowego obrobienia to np. kadencja, przyśpieszenia i przełożenia. Jak już wiadomo kadencja to ilość obrotów korby w ciągu minuty, a my mamy informację o ilości obrotów w ciągu sekundy. Sprawa prosta – dla aktualnej jednostki czasu wystarczy zsumować ostatnie 60sekund i wpisać w tablicę. Funkcja jest trochę bardziej sprecyzowana i nie uwzględnia momentów, gdzie prędkość wynosiła 0, czyli ktoś po prostu stał w miejscu. Poniższy kod przedstawia funkcję wyliczającą kadencję:

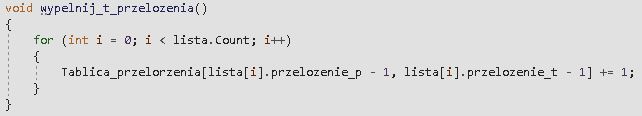


Funkcja została rozpisana dla odcinka czasu <60s i dla >=60s.

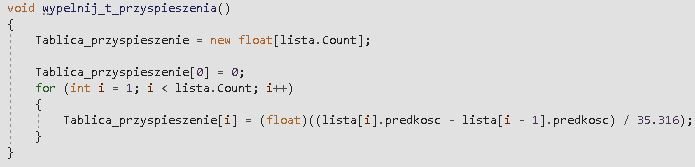
Informacje o przełożeniach natomiast są zbierane w dwuwymiarowej tablicy o wymiarach 3x9. [13]



Wykres z założenia miał przedstawiać względne użycie poszczególnych przełożeń więc stworzenie tablicy o wymiarach takich jak ilość przełożeń niesie za sobą dodatkowe korzyści. Prosta funkcja przeszukuje całą listę i dodaje po jednej sekundzie każdej z kombinacji biegów, np. dla przełożenia 2x5, funkcja zwiększa wartość komórki [1,4] o 1 (koordynaty o 1 mniejsze, gdyż zaczynamy numerację tablicy od 0). Dzięki takiej tablicy wiemy, które przełożenie jest naszym ulubionym oraz które blaty są używane najczęściej. Poniżej funkcja odpowiedzialna za wypełnienie tablicy:

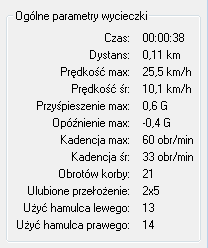


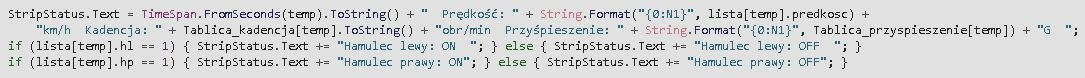
Tablica przyśpieszeń to już czysta fizyka a=(Vn+1-Vn)/(tn+1-tn). Na wytłumaczenie zasługuje tutaj liczba 35.316. Wynika ona z konwersji km/h na m/si tego, iż wynik chciałem przedstawić w postaci wartości względnej G przyśpieszenia ziemskiego. A więc mamy tutaj współczynnik z konwersji jednostek 3,6 oraz wartości przyśpieszenia ziemskiego g=9,81 co po przemnożeniu daje nam właśnie 35.316. Zakres danych tutaj będzie z zakresu raczej 0 – 1G, a więc wykorzystuję tutaj zmienną zmiennoprzecinkową float.

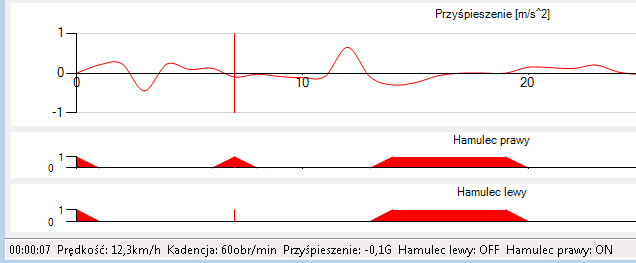


Kiedy już mamy przygotowane do wyświetlenia dane, możemy przystąpić do ich wyświetlenia. Używam tu funkcji dodawania serii punktów. Patrząc na pierwszy z góry wpis dotyczący prędkości, deklarujemy do jakiego wykresu dodajemy dane (chV), do jakiej serii danych („chV”), następnie następuje rzeczona funkcja dodania punktu i podajemy koordynaty X i Y punktu. Tak „przelatujemy” przez całą listę i dodajemy punkty z każdej sekundy wycieczki. [10][11]



W programie w prawej górnej części możemy również dostrzec wiele informacji dodatkowych, które można było wyliczyć/wywnioskować z posiadanych danych. Mamy tu np. wyliczony z prędkości, pokonany dystans, prędkość średnią i maksymalną, maksymalne przyśpieszenie i opóźnienie (podczas hamowania), maksymalną kadencję czy ciekawostki takie jak całkowita liczba zrobionych obrotów korbą podczas wycieczki czy użyć hamulców lewego i prawego. Są to parametry zebrane z całej wycieczki. Gdybyśmy jednak chcieli przeanalizować jakiś konkretny jej wycinek to mamy do dyspozycji kolejne narzędzie wskazujące wartości chwilowe. Na dole okna jest belka statusu (StripStatus) wyświetlająca właśnie wszystkie te informacje w miejscu znajdowania się kursora myszy. Dla lepszego zobrazowania, na wykresach pojawia się czerwona pionowa linia, wskazująca aktualnie mierzone miejsce. [12]





## Wymagania sprzętowe

Z aplikacji może skorzystać każdy posiadający komputer wyposażony w monitor, mysz, interface USB i system operacyjny Windows z zainstalowanymi bibliotekami .NET. Sama aplikacja zajmuje jedynie 1.31MB miejsca na dysku, a maksymalna wielkość pliku danych z programu to 160kB. Dodatkowo, aby móc zaimportować dane z urządzenia wymagane są sterowniki ze strony FTDI.

## Zalety używania aplikacji

Do czego można wykorzystać te wszystkie dane? Zalet jest wiele. Osoby aktywnie spędzające czas, przeprowadzające treningi mogą dokładnie prześledzić swój wysiłek. Są to parametry, które mierzą liczniki dla zawodowców. Drugą sprawą jest możliwość analizy wykorzystania sprzętu rowerowego. Gdybyśmy rejestrowali każdorazowo aktywność, moglibyśmy np. po ilości użyć hamulców wywnioskować za jaki czas będziemy mieli do wymiany okładziny hamulców. To samo tyczy się zębatek. Na podstawie ich użycia można przewidzieć która będzie najszybciej do wymiany. Na podstawie tych samych danych można zaplanować wymianę kasety (tylnych zębatek) i stwierdzić czy takie stopniowanie nam odpowiada, czy jednak warto wybrać inne. Na podstawie przebytych km można stwierdzić ile wytrzymały nasze opony i czy warto takie następnym razem kupować.

Źródła elektroniczne:

1. https://www.youtube.com/watch?v=I6uhMIFTF24&ab\_channel=CaturPebriandani
2. https://docs.microsoft.com/pl-pl/troubleshoot/developer/visualstudio/csharp/language-compilers/read-write-text-file
3. https://pl.wikipedia.org/wiki/USB
4. https://pl.wikipedia.org/wiki/Serial\_Peripheral\_Interface
5. https://pl.wikipedia.org/wiki/I%C2%B2C
6. https://pl.wikipedia.org/wiki/RS-232
7. https://wfis.uni.lodz.pl/kfcs/pdf/prup/embed/9Systemy\_wbudowane\_RS232.pdf
8. http://mikrokontrolery.blogspot.com/2011/03/rs-232-atmega8-komputer-terminal.html
9. https://www.youtube.com/watch?v=I8ZYsYrdeL4&ab\_channel=RJCodeAdvanceEN
10. https://www.youtube.com/watch?v=73KZAYJxRM4&ab\_channel=winforms
11. https://www.youtube.com/watch?v=g3bU16j-1WM&ab\_channel=ProgrammingGuru
12. https://www.youtube.com/watch?v=H0Kpx8Wafnw&ab\_channel=Krellon

Bibliografia:

1. Marcin Lis - C#. Praktyczny kurs. Wydanie III - 2016 – str. 98 Podstawowe operacje na tablicach
2. Ian Griffiths, Matthew Adams, Jesse Liberty - Programowanie C# - str.254 List<T>
3. Andrew Troelsen, Philip Japikse - Pro C# 7 - str. 141 Creating Structure Variables