Wiedza i doświadczenie projektowe wizytówką absolwenta kierunku automatyka i robotyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej

POKL.04.01.02-00-020/10

Program Operacyjny Kapitał Ludzki współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego

---

Gliwice, 27.II.2012r.

Międzywydziałowe Koło Naukowe High Flyers
Wydział Automatyki Elektroniki i Informatyki
Kierunek Automatyka i Robotyka

Raport z realizacji projektu:

Dobór i badania modułów nawigacji satelitarnej z przeznaczeniem do bezzałogowych platform latających.

Zespół projektowy:

Agnieszka Ziebura (Lider projektu)	
Andrzej Szmajnta	
Łukasz Szczurowski	
Michał Szuścik	
Oliver Kurgan	
Piotr Dyga	
Michał Ożga (mentor projektu)	
Podpis opiekuna koła naukowego	

1. Opis projektu - co zrealizowano:

1.1. **Cel projektu:** Zaprojektowanie, wykonanie, implementacja i testowanie modułowych rozwiązań programowych oraz sprzętowych wykorzystujących technologię GPS.

Zaprojektowano, wykonano i przetestowano część programową oraz wstępnie zaprojektowano część sprzętową.

1.2. **Założenia projektu:** Założeniem projektu jest zbudowanie uniwersalnych modułów korzystających z technologii GPS umożliwiających wykorzystanie nawigacji satelitarnej w projektach koła naukowego.

Założenie projektu, czyli zbudowanie uniwersalnych modułów korzystających z technologii GPS, nie zostało w pełni zrealizowane. Moduły zostały sprawdzone oraz przetestowane na zakupionych płytach ewaluacyjnych wyposażonych w mikrokontrolery AVR.

1.3. **Oczekiwane wyniki:** Wynikiem przeprowadzonego projektu ma być uniwersalny moduł sprzętowo-programowy będący podstawą dla testowania algorytmów nawigacyjnych opartych o system GPS. Moduł pozwalać będzie również na jego bezpośrednią implementacje w innych projektach koła naukowego.

Oczekiwane wyniki projektu nie zostały w pełni osiągnięte. W całości wykonano zadanie pierwsze projektu: **GPS dla PC**. Wykonane zostało również zadanie drugie: **GPS dla AVR**. Stworzenie uniwersalnego modułu wyposażonego w mikrokontroler AVR i moduł GPS byłoby odwzorowaniem modelu złożonego w oparciu o płytę ewaluacyjną AVR EvB 4.3 v4, jednak zabrakło nam czasu na wykonanie płytek PCB.

1.4. **Ocena ryzyka projektu:** Zakładając realizację projektu przez osoby o różnym stopniu zaznajomienia z układami elektronicznymi istnieje ryzyko związane z uszkodzeniem niektórych elementów elektronicznych. Uwzględniając natomiast dotychczasowe zaangażowanie uczestników projektu w ramach prac w strukturach koła naukowego, ich ogólną chęć poszerzania wiedzy, dodając do tego ścisłą współpracę z opiekunami naukowymi, można określić ryzyko niezrealizowania celów projektowych na minimalne.

We wniosku o realizację projektu w ocenie ryzyka projektu nie uwzględniono faktu, iż prace zespołu projektowego nad zadaniami mogą zostać zahamowane z powodu niedostępności niezbędnego sprzętu w odpowiednim czasie, który miał zostać wykorzystany do realizacji projektu. W związku z tym prace nad drugim zadaniem projektu mogliśmy rozpocząć dopiero po przerwie międzysemestralnej i jak dotąd, trwały one niewystarczająco długo, aby sfinalizować projekt.

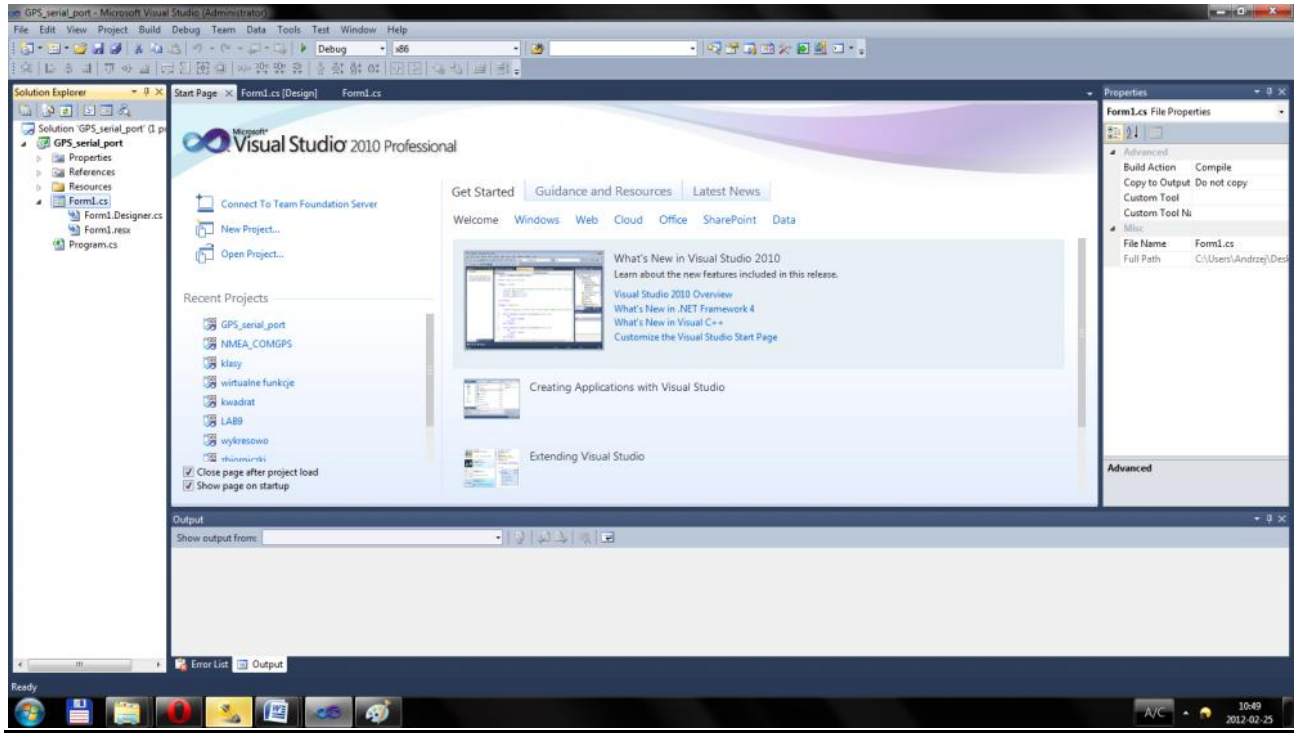
2. Podział projektu na zadania – opis ich realizacji:

1. GPS dla PC.

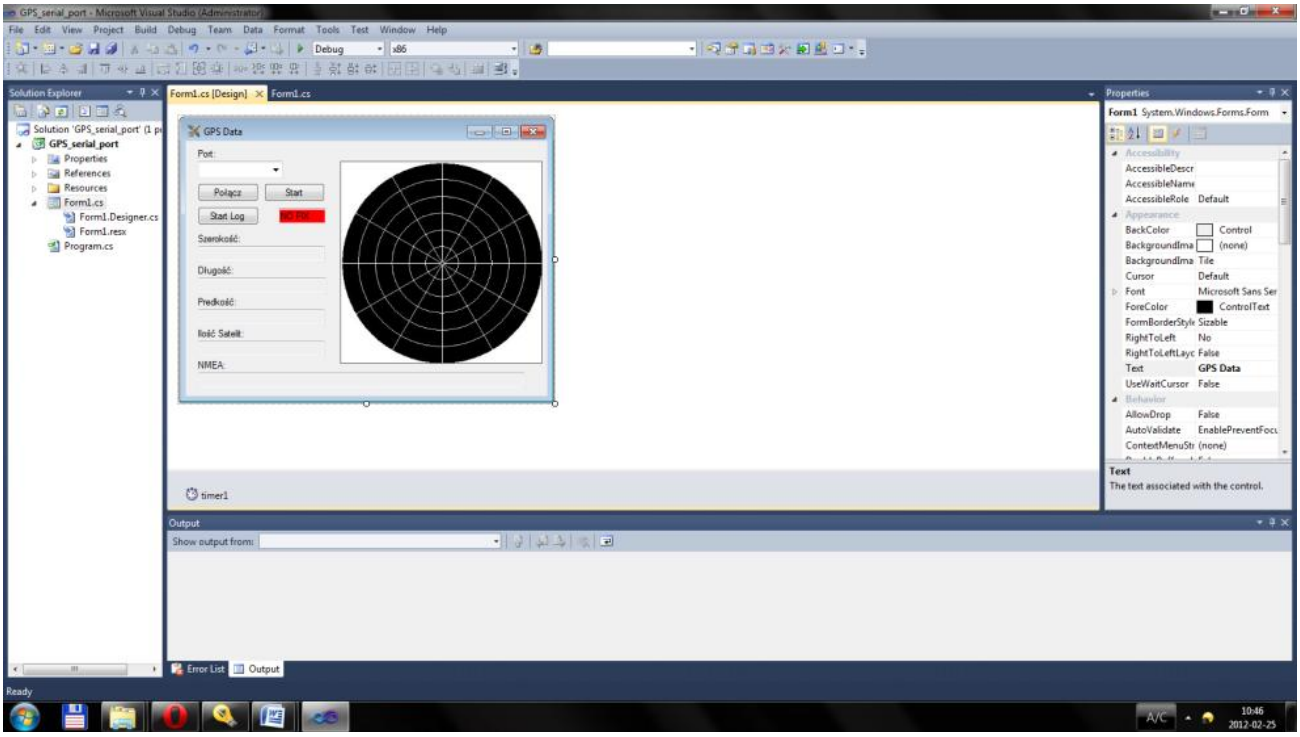
1.1. Wybór języka oraz środowiska programistycznego.

Jako język programowania wybrany został język C#, ponieważ bardzo dobrze sprawdza się w przypadku pisania aplikacji okienkowych. Ze względu na wcześniejszą znajomość i doświadczenie w pracy z językiem C oraz C++, opanowanie C# w zakresie pozwalającym na napisanie funkcjonalnej aplikacji nie stanowiło większych problemów.
Jako środowisko programistyczne wybraliśmy produkt firmy Microsoft, a mianowicie środowisko programistyczne RAD o nazwie VisualStudio 2010. W ramach uczelni środowisko jest dostępne w programie MSDNAA. Program posiada wbudowane liczne biblioteki, upraszczające i przyspieszające proces pisania programów.

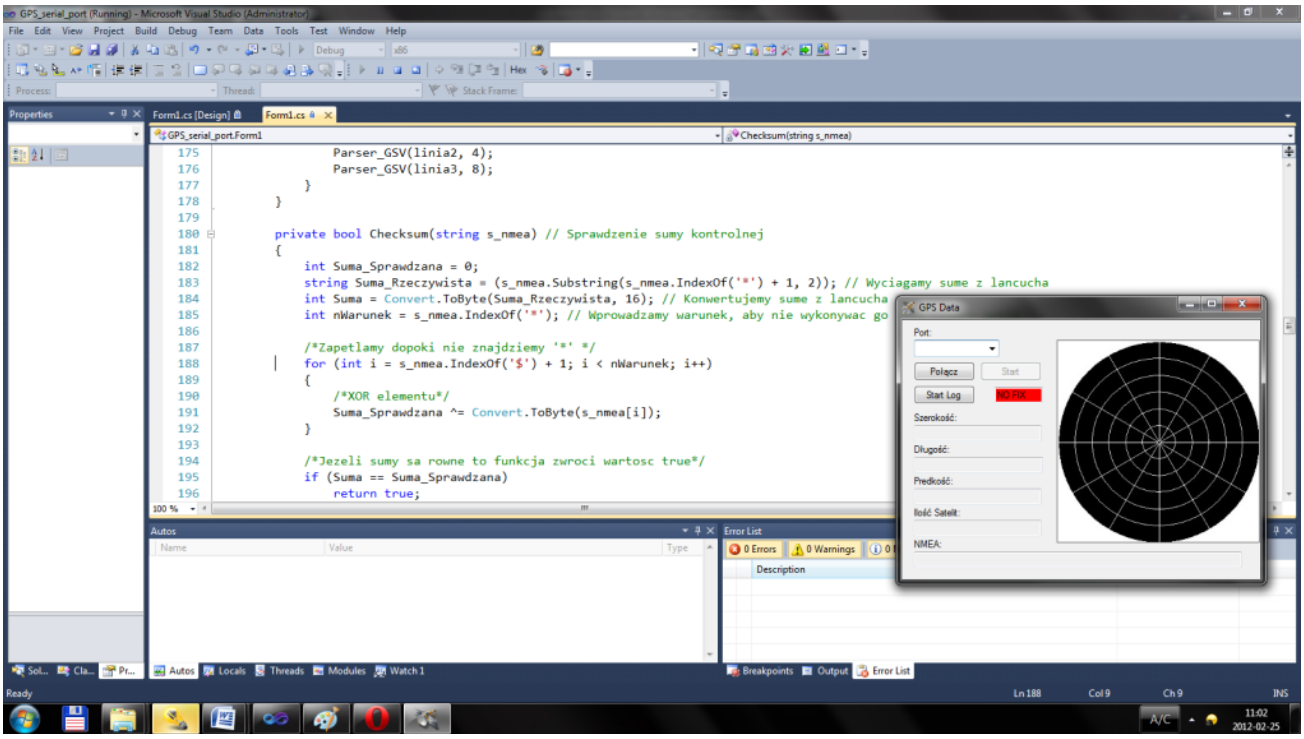
<u>Możliwości języka C# i środowiska VisualStudio:</u>



Rysunek 1.1.1. *Wygląd strony startowej Microsoft VisualStudio2010.*



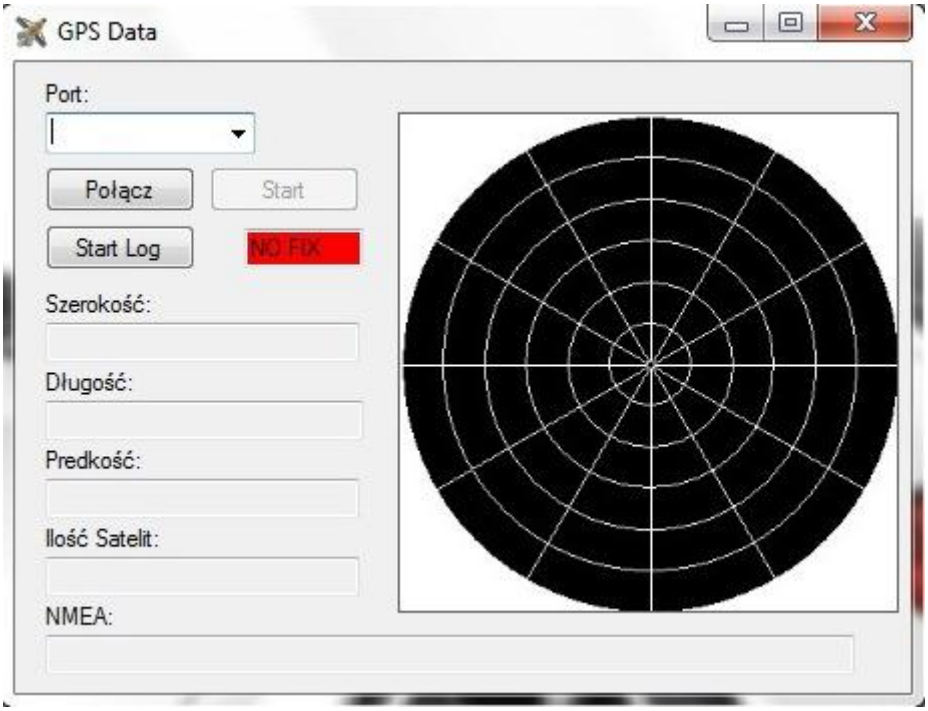
Rysunek 1.1.2. *Projektowanie wyglądu "okienka".*



Rysunek 1.1.3. *Fragment kodu C# w Visual Studio 2010 razem z włączoną aplikacją.*

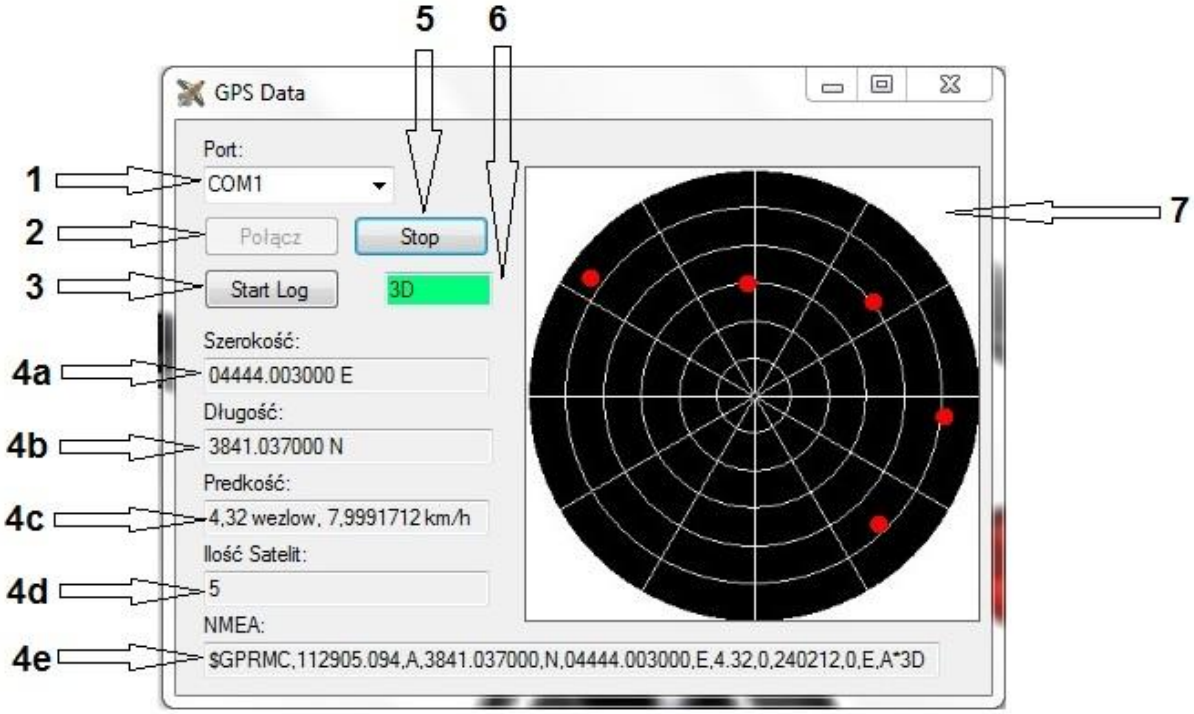
1.2. Zaprojektowanie graficznego interfejsu użytkownika.

Stworzony przez nas graficzny interfejs użytkownika umożliwia interakcję programu z użytkownikiem. Staraliśmy się, by miał on jak najbardziej przystępną i przejrzystą formę, a przyciski funkcyjne oraz informacje przesyłane przez moduł GPS były rozmieszczone w interfejsie w intuicyjny sposób. Chcieliśmy, by aplikacja była przyjazna dla każdego, nawet początkującego użytkownika.



Rysunek 1.2.1. *Widok graficznego interfejsu użytkownika przy wyłączonym programie.*

Nasz interfejs pozwala na wybór portu, przez który połączymy się z modułem GPS. Ponadto po uruchomieniu aplikacji i połączeniu z modułem GPS, można zażądać, by program logował do pliku dane dotyczące aktualnego położenia. Informacje, które odczytuje program: aktualna szerokość i długość geograficzna oraz prędkość, także ilość aktualnych połączeń z satelitami. W interfejsie wyświetlany jest również nieprzetworzony ciąg NMEA, informacja o FIX-ie GPS oraz mapka pokazująca rozmieszczenie satelit.



Rysunek 1.2.2. *Aplikacja podczas pracy.*

<u>Opis poszczególnych elementów interfejsu:</u>

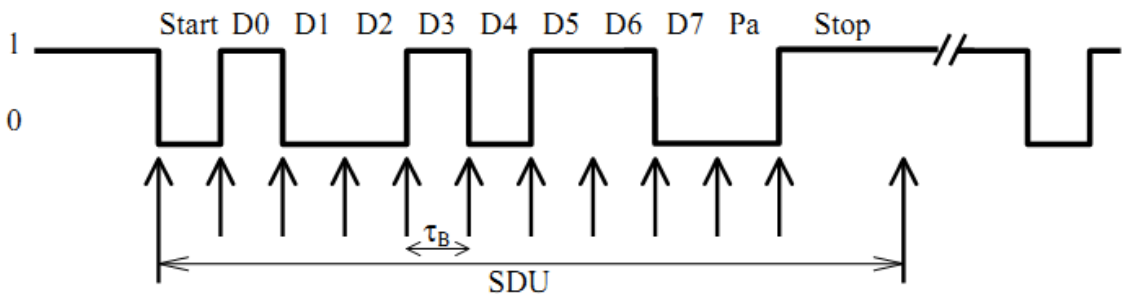
1) Wybór portu do połączenia z modułem GPS.
2) Przycisk inicjujący połączenie.
3) Przycisk inicjujący logowanie do pliku danych o położeniu.
4) Informacje przesyłane przez moduł GPS:
   a) aktualna szerokość geograficzna,
   b) aktualna długość geograficzna,
   c) aktualna prędkość,
   d) ilość aktualnych połączeń z satelitami,
   e) nieprzetworzony ciąg NMEA.
5) Przycisk zatrzymujący działanie aplikacji.
6) Informacje o FIX-ie GPS.
7) Mapka pokazująca rozmieszczenie satelit.

1.3. Obsługa portu komunikacyjnego (szeregowego).

Standard komunikacyjny zaproponowany przez NMEA – protokół wykorzystywany do komunikacji z odbiornikiem GPS – musi być zgodny z EIA-422. Do komunikacji z urządzeniami GPS możemy użyć zwykłego portu COM obecnego w komputerze. Ważne jest ustawienie interfejsu zgodnie z przyjętym standardem:

- prędkość – 4800 bodów,
- 8 bitów danych,
- brak kontroli parzystości,
- 1 bit stopu.

Interfejs RS232 służy do dwukierunkowej synchronicznej lub asynchronicznej transmisji danych w sposób szeregowy. Określa on połączenie między terminalem DTE (*Data Terminal Equipment*) a sprzętem komunikacyjnym DCE (*Data Communication Equipment*). W naszym przypadku rolę terminalu spełnia komputer PC, a sprzętu komunikacyjnego – moduł GPS. Układem elektronicznym realizującym taki przesył danych jest USART (ang. *Universal Synchronous and Asynchronous Receiver and Transmitter*). Zawiera on konwerter równoległo-szeregowy (ang. *parallel-to-serial*) służący do konwersji danych na postać szeregową oraz konwerter szeregowo-równoległego (ang. *serial-to-parallel*) do konwersji odwrotnej. Układ ten zawiera także zazwyczaj bufory danych odbieranych i wysyłanych. Obecnie produkowane mikrokontrolery zawierają co najmniej jeden moduł transmisji USART.
Przesyłanie informacji za pomocą USART następuje w sposób szeregowy bit po bicie. Stany logiczne 0 i 1 kodowane są stanami napięć. Jednostkę informacyjną SDU (ang. *Serial Data Unit*) tworzą bit startu, 5 do 8 bitów danych informacyjnych, opcjonalnego bitu kontroli parzystości oraz od 1 do 2 bitów STOP.



Rysunek 1.3.1. *Przykładowa ramka danych.*

Bity danych są przesyłane w kolejności od najmniej znaczącego D0, do najwięcej znaczącego. Opcjonalny bit parzystości ma wartość logiczną równą sumie modulo 2 wszystkich bitów danych. Powyższy rysunek przedstawia przykładową ramkę danych.
Standard RS-232 opisuje sposób podłączeń, nazwy styków złączy, sygnałów a także specyfikację elektryczną obwodów wewnętrznych. Specyfikacja elektryczna standardu RS232 definiuje "1" logiczną jako napięcie od -3V do -15V, zaś "0" to napięcie od +3V do +15V. Poziom napięcia wyjściowego natomiast może przyjmować wartości -12V, -10V, +10V, +12V, zaś napięcie na dowolnym styku nie może być większe niż +25V i mniejsze niż -25V.

Obsługa portu w C# :

```
SerialPort mySerialPort = new SerialPort("COM1");
        mySerialPort.BaudRate = 9600;
        mySerialPort.Parity = Parity.None;
        mySerialPort.StopBits = StopBits.One;
        mySerialPort.DataBits = 8;
        mySerialPort.Handshake = Handshake.None;
        mySerialPort.Open();
        mySerialPort.Close();
```

1.4. Parser GPS.

Parser GPS jest niezbędny do wydobycia odpowiednich informacji z danych przesyłanych przez moduł GPS. Nasz program zawiera taki specjalnie napisany parser, który wychwytuje interesujące nas informacje.
Z uwagi na to, iż z odbiornika GPS możemy odczytać nie tylko pozycję użytkownika (długość geograficzną, szerokość geograficzną, wysokość), lecz również dodatkowe informacje dotyczące aktualnego czasu (satelity GPS mają kilka zegarów atomowych), mocy odbieranych sygnałów z satelitów, ich aktualnej pozycji na orbicie około ziemskiej, ilości widocznych w danym momencie satelitów na których podstawie wyznaczony był pomiar, błędy pomiaru oraz inne dodatkowe informacje, do komunikacji z odbiornikiem GPS wykorzystywany jest specjalny protokół NMEA(National Marine Electronics Association's), który jest najczęściej używanym standardem w większości urządzeń i systemów nawigacyjnych. Organizacja NMEA stworzyła jednoznaczną specyfikację interfejsu komunikacyjnego i opis protokołu, który umożliwia komunikację między różnego rodzaju urządzeniami pomiarowymi i prostą integrację zakupionego modułu GPS z innymi urządzeniami. Standard cały czas ewoluuje, gdyż tworzone są coraz to nowe urządzenia. Aktualnie obowiązującym numerem standardu jest 3.01. Jednakże wszystkie późniejsze zmiany są jedynie dodatkami do wyjściowego (i najpowszechniej stosowanego obecnie) standardu 0183 (wersja 2.0) (NMEA-0183). Każdy odbiornik GPS obsługujący nowszy standard musi być również zgodny ze standardem NMEA-0183.
Dane w NMEA zawsze zaczynają się od znaku ‘$’, po którym występuje dwuliterowy identyfikator urządzenia nadającego oraz trzyliterowe hasło kluczowe - nagłówek. Po nagłówku następuje miejsce na przekazywane informacje. Każda informacja ma swoje konkretne miejsce w zdaniu, a kolejne dane oddziela się od siebie przecinkami. Na końcu zdania występuje znak ‘*’ oraz liczba kontrolna w postaci heksadecymalnej, która pozwala sprawdzić czy podczas przesyłania nie uległy przekłamaniu i ewentualnie odrzucić daną linię. Dane wysyłane są w sposób tekstowy, a każda linia danych jest niezależna od innych. Ustalenia te sprawiają, iż jest to standard bardzo uniwersalny. Przykładowy ciąg danych NMEA:

Standardowe sekwencje nagłówków opisane w standardzie NMEA-0183 (wszystkie rozpoczynają się od znaków GP) są następujące:

- AAM – Waypoint Arrival Alarm,
- ALM – Almanac data,
- APA – Auto Pilot A sentence,
- APB – Auto Pilot B sentence,
- BOD – Bearing Origin to Destination,
- BWC – Bearing using Great Circle route,
- DTM – Datum being used,

- GGA – Fix information,
- GLL – Lat/Lon data,
- GSA – Overall Satellite data,
- GSV – Detailed Satellite data,
- MSK – Send control for a beacon receiver,
- MSS – Beacon receiver status information,
- RMA – Recommended Loran data,
- RMB – Recommended navigation data for gps,
- RMC – Recommended minimum data for gps,
- RTE – Route message,
- VTG – Vector track an Speed over the Ground,
- WCV – Waypoint closure velocity (Velocity Made Good),
- WPL – Waypoint information,
- XTC – Cross track error,
- XTE – Measured cross track error,
- ZTG – Zulu (UTC) time and time to go (to destination),
- ZDA – Date and Time.

W parserze wykorzystuje się ramkę o słowie kluczowym RMC (Recommended Minimum sentence), która jest najczęściej używanym zdaniem przekazywanym przez GPS. To jedno zdanie posiada niemal wszystkie najważniejsze dane na temat pozycji obiektu. Format ramki przedstawia się następująco:

```
$GPRMC,HHMMSS.SS,A,DDMM.MMM,N,DDDMM.MMM,W,Z.Z,Y.Y,DDMMYY,D.D,V*CC<CR><LF>
```

Oznaczenie	**Opis**
HHMMSS.SS	Czas UTC
A	Poprawność danych (A – poprawne, V – niepoprawne)
DDMM.MMM	Szerokość geograficzna w stopniach, minutach i minutach w formie dziesiętnej
N	Lokalizacja szerokości (N – Szerokość północna S – Szerokość południowa)
DDDMM.MMM	Długość geograficzna w stopniach, minutach i minutach w formie dziesiętnej
W	Lokalizacja długości (E – Długość wschodnia, W – Długość zachodnia)
Z.Z	Prędkość ciała podana w węzłach
Y.Y	Kierunek względem powierzchni ziemi podany w stopniach
DDMMYY	Data (Dzień, Miesiąc, Rok)
D.D	Kierunek magnetyczny podany w stopniach
V	Opis kierunku magnetycznego (E – wschodni, W – zachodni)
*CC	Suma kontrolna
<CR>	Carriage Return
<LF>	Line Feed

Tabela 1.4.1. *Opis danych znajdujących się w ramce o słowie kluczowym RMC.*

Ważnym elementem pracy GPS jest określenie, czy odbiornik posiada tzw. fixa, tzn. czy odpowiednio odbiera sygnał z satelit. Informację o tym można znaleźć w innej ramce – GSA, na miejscu drugim. Dla przykładu podkreślono informację o fixie w poniższej ramce:

`$GPGSA,A,3,19,28,14,18,27,22,31,39,,,,,1.7,1.0,1.3*34`

W miejscu tym można otrzymać trzy różne informacje:

1 – Oznacza, że FIX jest nieosiągalny,
2 – Oznacza, że FIX jest 2D,
3 – Oznacza, że FIX jest 3D.

W napisanym programie, specjalnie stworzony parser wyszukuje tę informację i podaje ją na ekran użytkownika.

Algorytm wykorzystywany przy wydobywaniu informacji jest następujący:

1. Program odczytuje otrzymaną linię od GPSa i sprawdza jego sumę kontrolną. Jeżeli suma kontrolna się nie zgadza, to program odczytuje kolejną linię.
2. Program określa, jakiego rodzaju (RMC/GSV/GSA) jest ramka i kieruje otrzymany łańcuch znaków do odpowiedniej funkcji parsującej. Jeżeli program odczytał linię o innej ramce niż podana, to wraca do punktu 1.
3. Odpowiednia funkcja otrzymuje łańcuch znaków i tworzy zmienne tymczasowe typu double, pod którymi będzie zapisywać wyłuskane dane.
4. Program wykonuje pętlę, która analizuje znak po znaku otrzymany łańcuch. Pętla kończy się w momencie znalezienia znaku '*'.
5. Pętla wyszukuje i zlicza występowanie kolejnych znaków oddzielających ','. Jeżeli pętla natrafi na interesujący ją przedział znaków między przecinkami, to zapisuje kolejne napotkane znaki do zmiennej tymczasowej typu string. Program stworzy w ten sposób z pełnego łańcucha kilka mniejszych łańcuchów znaków zawierających poszukiwane przez nas dane.
6. Po wykonaniu się całej pętli program zamienia wyodrębnione łańcuchy znaków typu string na liczby zmiennoprzecinkowe typu double za pomocą klasy *Convert.ToDouble*.
7. Program zamienia otrzymane wartości na interesujące nas jednostki (np. przelicza prędkość z węzłów na km/h).
8. Program wypisuje otrzymane dane na ekran użytkownika. Jeżeli użytkownik sobie tego życzy, zapisuje wyłuskane wartości w dataloggerze.

1.5. Zapewnienie prostej wielowątkowości aplikacji.

Wielowątkowość realizuje wykonywanie kilku wątków lub jednostek wykonawczych w ramach jednego procesu. Nowe wątki to kolejne ciągi instrukcji wykonywane oddzielnie. Wszystkie wątki tego samego procesu współdzielą kod programu i dane.

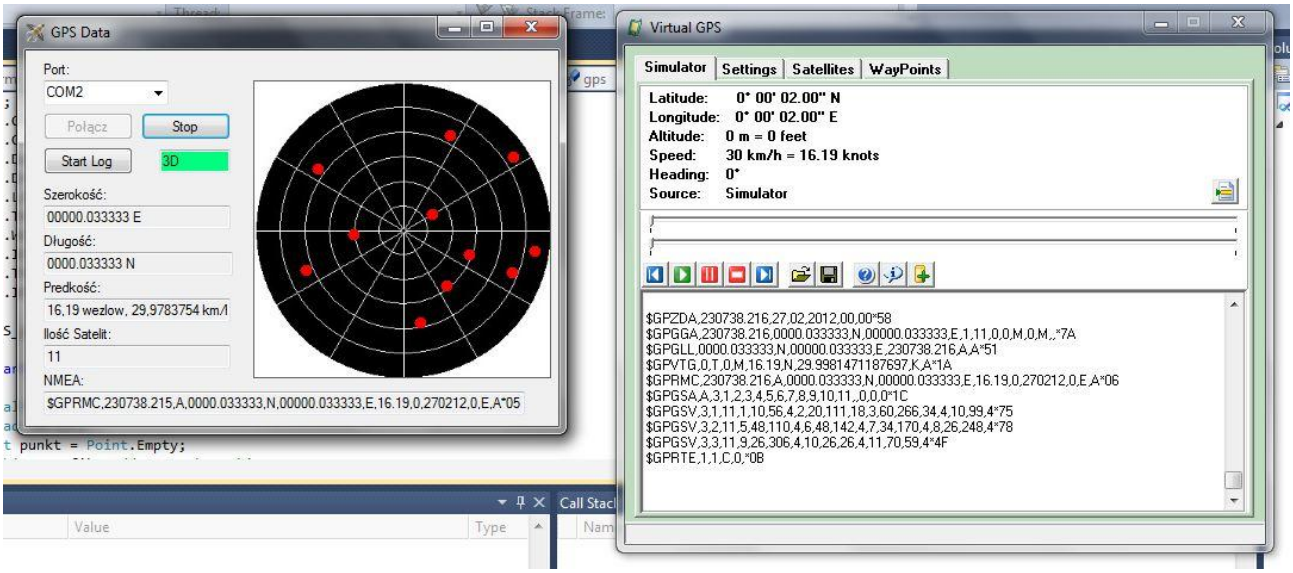
Cechy wielowątkowości:

- wszystkie wątki wykonują się w ramach tylko jednego programu (procesu). Innymi słowy, jeden proces posiada wiele instancji wykonawczych (wątków),
- wprowadzenie wątków może obniżyć wydajność ponieważ, najczęściej wymagane jest przy tym wprowadzenie odpowiednich mechanizmów synchronizacji,
- wszystkie wątki procesu współdzielą tę samą wirtualna przestrzeń adresową (mają dostęp do tych samych zmiennych, obiektów i struktur) i korzystają z tych samych zasobów systemowych,
- komunikacja między wątkami, w odróżnieniu od procesów, jest bardzo łatwa do wykonania: wystarczy odwoływać się do tych samych zmiennych i obiektów,
- współdzielenie wirtualnej przestrzeni adresowej niesie zagrożenie – jeden "wadliwy" wątek może zagrozić wykonaniu całego programu.

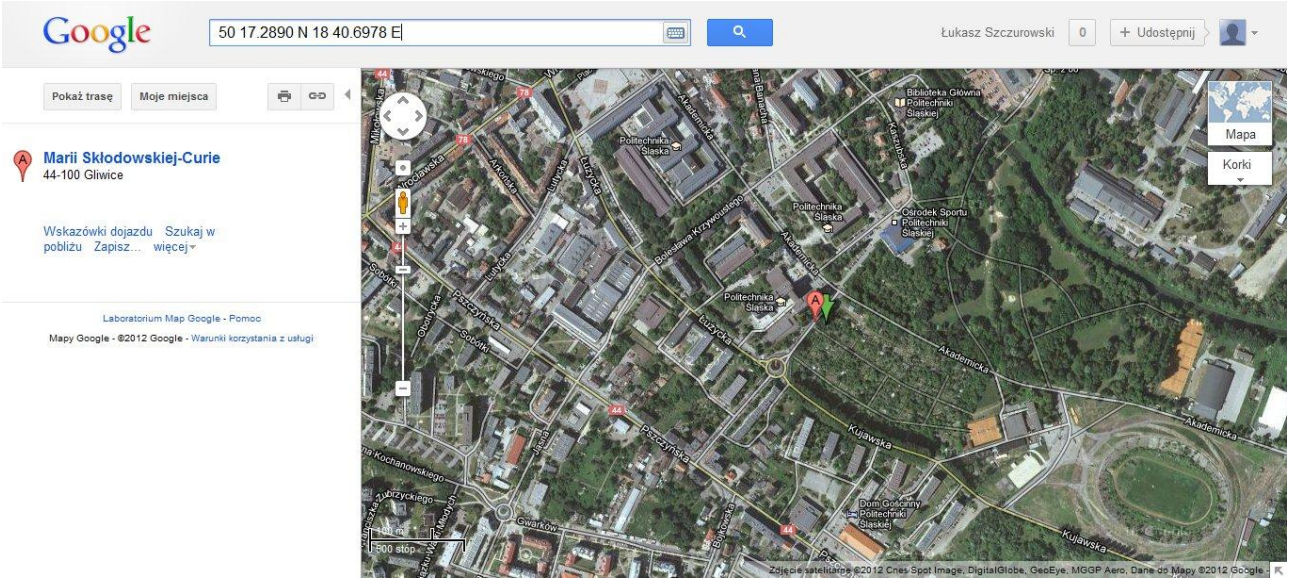
W naszym programie korzystaliśmy z przestrzeni nazw **System.Threading**, a podczas tworzenia właściwych wątków – z klasy **Thread**. Dzięki wielowątkowości nasz program działa szybciej i sprawniej, udało nam się także wyeliminować kilka błędów mających miejsce podczas pobierania i wyświetlania informacji z modułu GPS.

1.6. Testy aplikacji.

Testy napisanej aplikacji, zarówno przy wykorzystaniu symulatora GPS, jak i sprzętu rzeczywistego – zewnętrzny GPS bluetooth, wypadły pomyślnie.



Rysunek 1.6.1. *Obrazuje pomyślne wyniki testów aplikacji na symulatorze.*

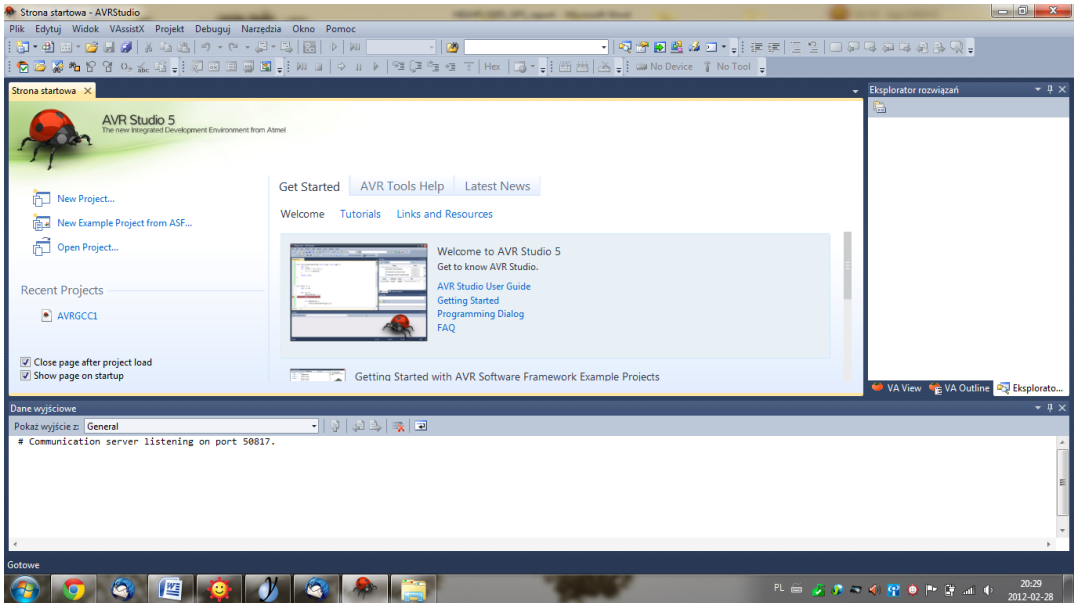


Rysunek 1.6.2. *Obrazuje pomyślne wyniki testów aplikacji na sprzęcie rzeczywistym.*

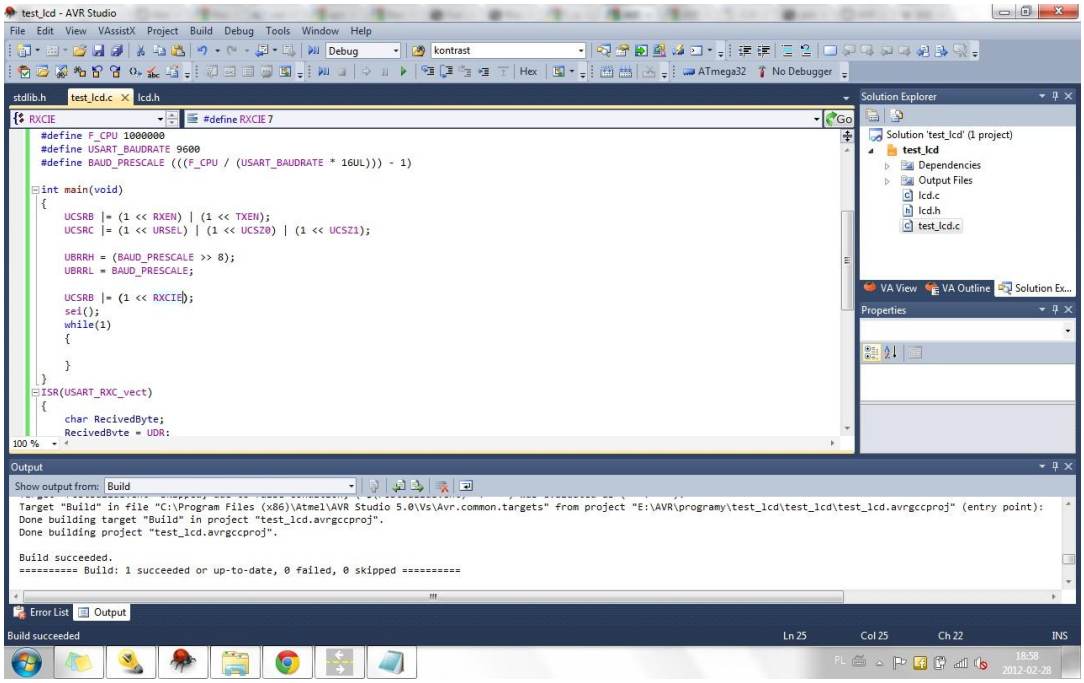
2. GPS dla AVR.

2.1. Wybór języka i środowiska programistycznego dla mikrokontrolerów AVR.

Jako język programowania mikrokontrolerów AVR wybraliśmy język C. Główną przyczyną była jego wcześniejsza znajomość i zastosowanie w pracy języka. Poza tym, język C jest jednym z najstarszych i najszerzej stosowanych języków programowania. Jest językiem niskiego poziomu, dzięki czemu programista ma łatwy dostęp do zasobów sprzętowych, co jest szczególnie przydatne w przypadku programowania mikrokontrolerów. Równocześnie jest wyższego poziomu niż asembler, dzięki czemu programowanie jest o wiele łatwiejsze.
Wybranym środowiskiem programistycznym jest AVR Studnio – produkt firmy Atmel, będącej jednocześnie producentem mikrokontrolerów AVR. Postanowiliśmy go użyć, ponieważ cechuje go maksymalna zgodność z architekturą rodziny AVR.



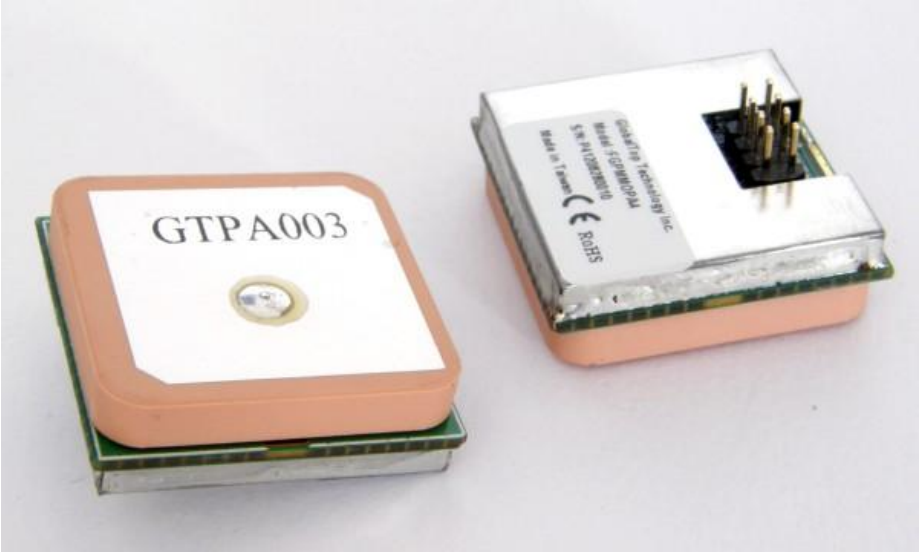
Rysunek 2.1.1. *Okno startowe AVR Studio.*



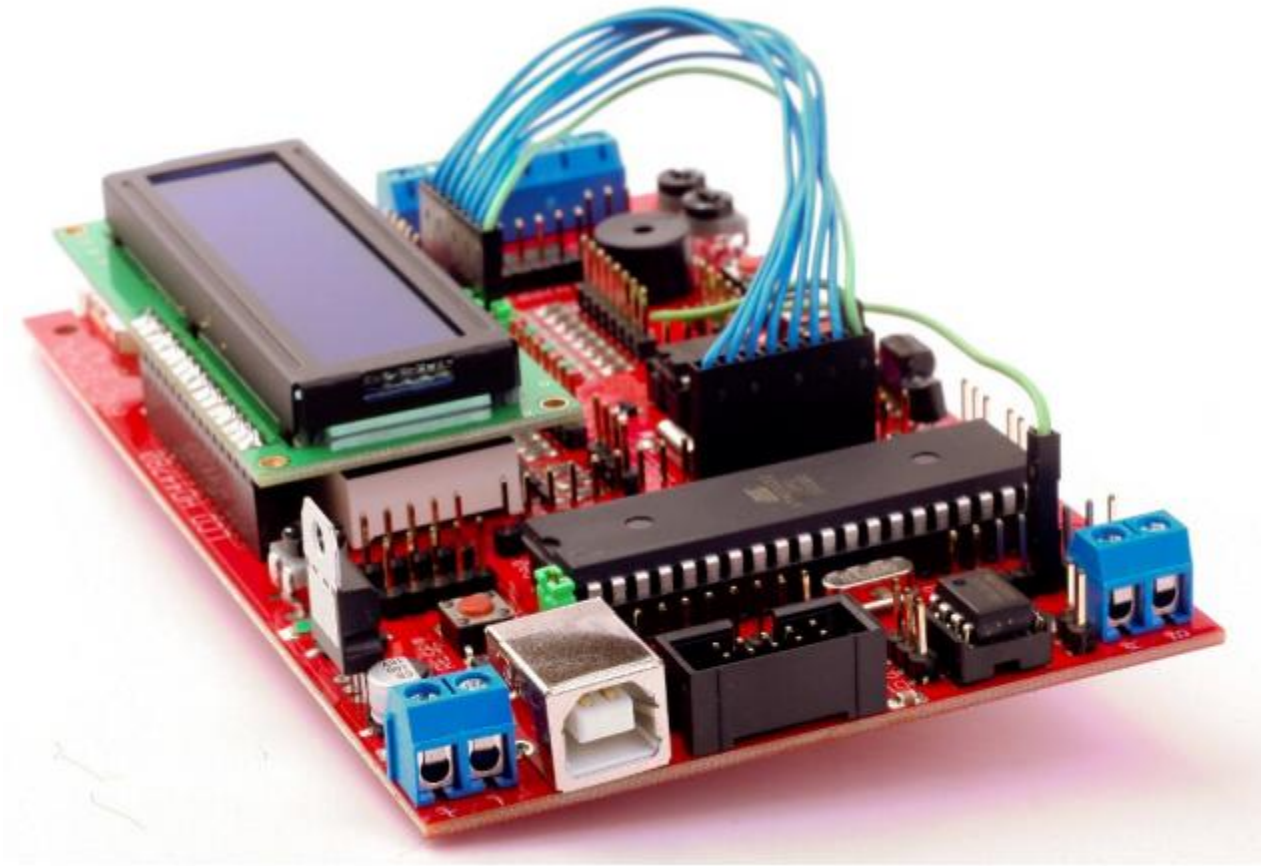
Rysunek 2.1.2. *Fragment kodu C w AVR Studio.*

2.2. Projekt platformy sprzętowej, wybór elementów.

Wybrany został 8-mio bitowy mikrokontroler AVR z rodziny Atmega. Jako odbiornik GPS wybrano GPS-FGPMMOPA4, ponieważ jest dokładny, ma niewielkie rozmiary, wyprowadzenia na goldpiny oraz wbudowaną wewnętrzną antenę.



Rysunek 2.2.1. *GPS-FGPMMOPA4.*



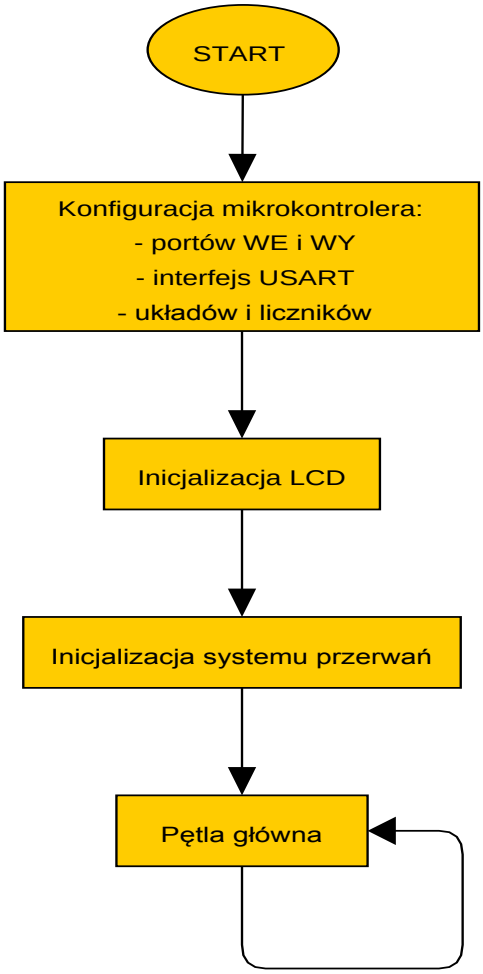
Rysunek 2.2.2. *Płytka uruchomieniowa EvB 4.3 v4.*

2.3. Określenie założeń funkcjonalnych oprogramowania dla mikrokontrolera AVR.

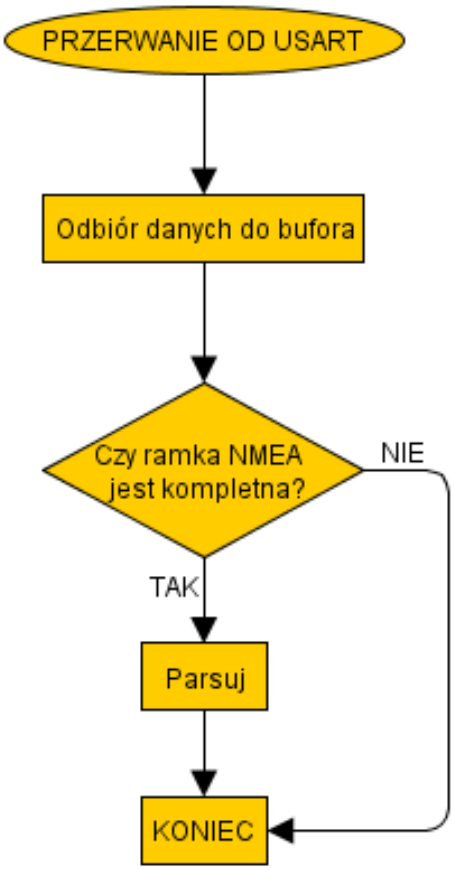
Oprogramowanie:

- umożliwia podejrzenie aktualnych danych nawigacyjnych na dołączonym wyświetlaczu LCD,
- obsługuje interfejs użytkownika opierający się na wyświetlaczu i przyciskach funkcyjnych,
- działa w sposób asynchroniczny – wykorzystano system przerwań mikrokontrolera.

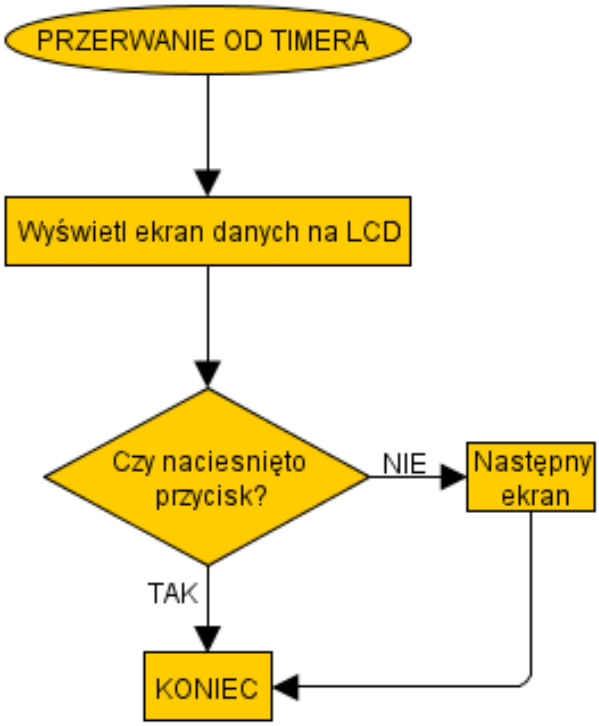
2.4. Utworzenie szkieletu oprogramowania i obsługa mikrokontrolera AVR.



Rysunek 2.4.1. *Główny schemat blokowy programu dla mikrokontrolera AVR.*



Rysunek 2.4.2. *Obsługa przerwania portu szeregowego.*



Rysunek 2.4.3. *Obsługa przerwania układu licznikowego (timer'a).*

2.5. Obsługa wyświetlacza LCD i przycisków sterujących (AVR).

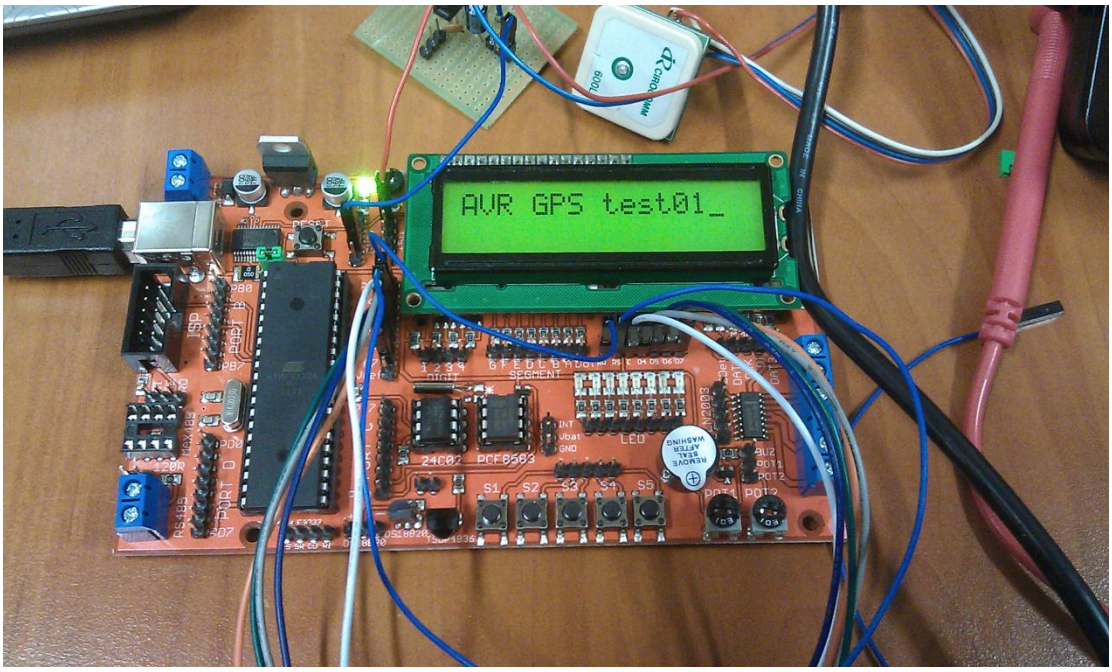
Obsłużono wyświetlacz LCD alfanumeryczny w trybie czterobitowym.

2.6. Obsługa portu komunikacyjnego UART (AVR).

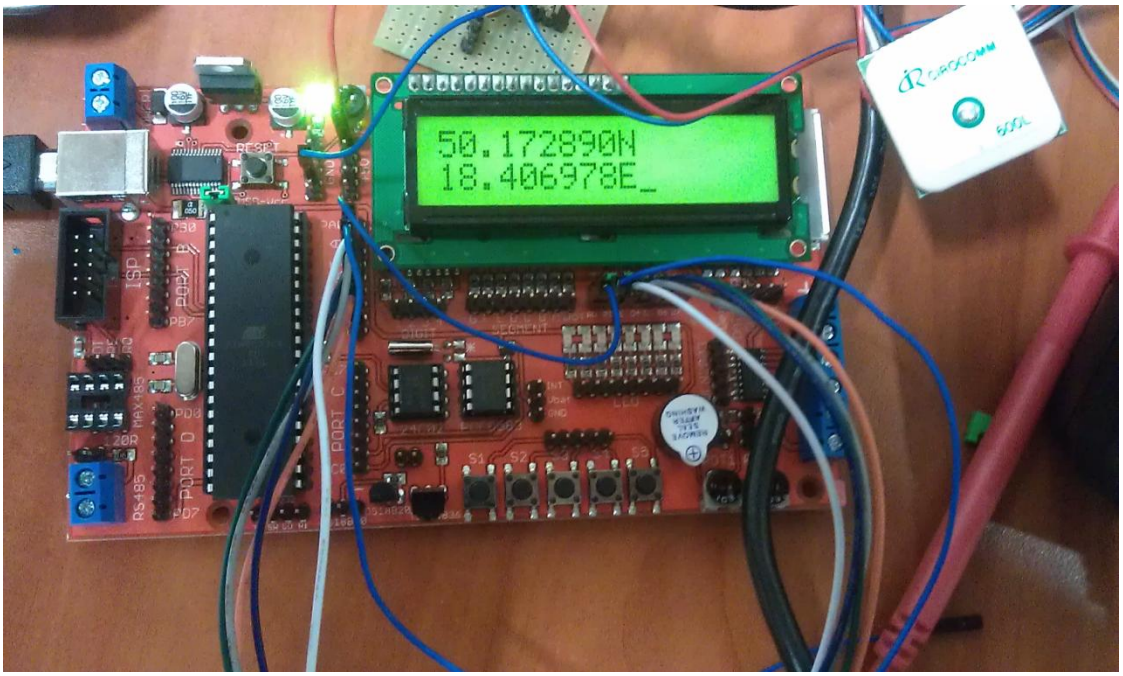
Obsłużono port UART wykorzystując system przerwań mikrokontrolera.

2.7. Obsługa odbiornika GPS – Parser GPS (AVR).

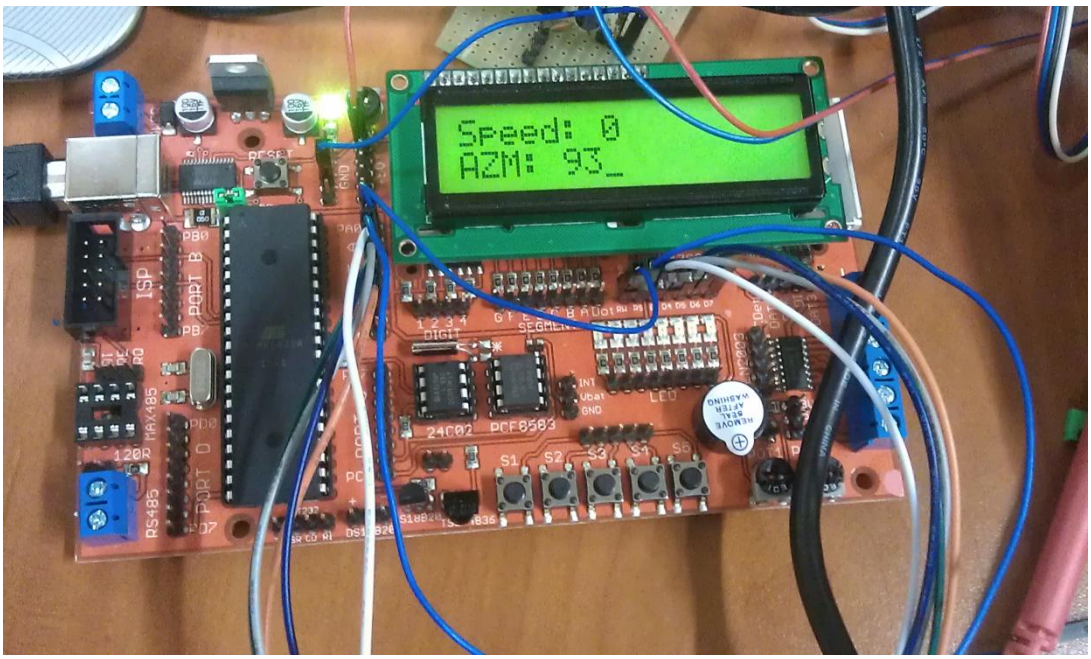
Obsłużono odbiornik GPS poprzez port UART.



Rysunek 2.7.1. *Test wyświetlacza.*



Rysunek 2.7.2. *Badania odbiornika GPS - aktualna pozycja.*



Rysunek 2.7.3. *Doświadczenia z odbiornikiem GPS - wyświetlanie informacji o prędkości i kierunku poruszania się.*

3. Nawigator GPS-AVR

3.1. Określenie założeń funkcjonalnych.

- określanie kierunku do zdefiniowanego wcześniej weypoint'a,
- wyświetlanie aktualnego kierunku poruszania się (azymutu),
- wyświetlanie azymutu punktu docelowego,
- wyświetlanie odległości do punktu docelowego,
- wyświetlanie prostych komend słownych.

3.2. Opracowanie i analiza algorytmów nawigacyjnych.

Algorytmy nawigacyjne zostały jedynie zaproponowane ideowo. Ze względu na brak czasu nie udało się opracować jednolitego algorytmu.

3.3. Implementacja i testowanie.

W związku z problemami w realizacji podpunktu 3.2, nie rozpoczęliśmy etapu implementacji i testowania algorytmów nawigacyjnych.

4. Datalogger GPS-AVR

4.1. Określenie założeń funkcjonalnych i dobór rozwiązań sprzętowych.

- Wykonanie dataloggera opartego na urządzeniu przenośnym klasy PC.
- Założenia funkcjonalne dla dataloggera opartego na zaprojektowanej platformie sprzętowej (AVR):
  a. zapisywanie pomiaru położenia z częstotliwością 0,5 Hz,
  b. możliwość konwersji pomiarów do formatu akceptowalnego przez Google Earth,
  c. możliwość przedstawienia przebytej trasy w programie Google Earth,
  d. wyświetlanie informacji o położeniu w czasie rzeczywistym na ekranie LCD.

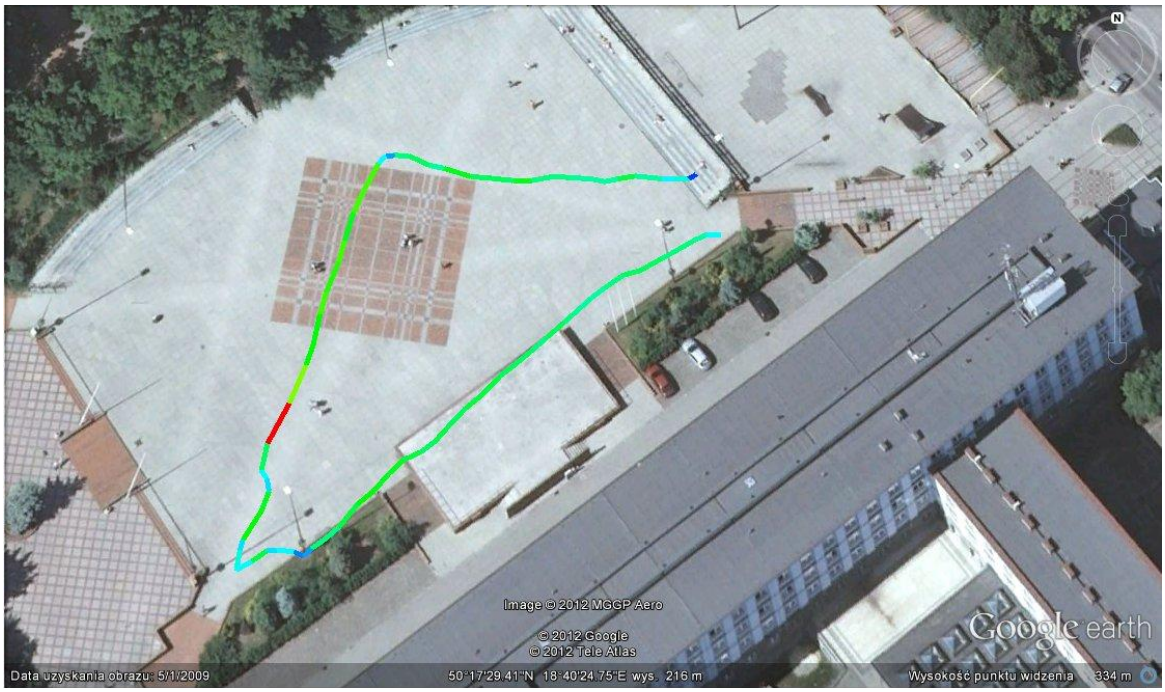
4.2. Obsługa zewnętrznego nośnika pamięci.

Dane zapisywane były w pliku na zewnętrznym nośniku (karta SD).

4.3. Datalogger dla zaprojektowanej platformy sprzętowej (AVR).

Ze względu na brak czasu i doświadczenia nie udało nam się zrealizować tego podpunktu w całości. Zdążyliśmy jedynie wykonać analogiczny datalogger w oparciu o urządzenie przenośne klasy PC.
Realizacja podpunktu:

Rysunek 4.3.1. *Wyniki testów dataloggera opartego na PC.*



Rysunek 4.3.2. *Wyniki testów dataloggera opartego na PC.*

Wyniki przeprowadzonych testów nawigatora GPS-AVR przedstawiają powyższe fotografie. Wykonaliśmy je przy pomocy programu Google Earth, do którego wczytaliśmy dane zebrane przez datalogger. Trasy testowe wykonaliśmy na Placu Krakowskim, założeniem było wykonanie trasy po obwodzie placu (Rysunek 3.3.1), a następnie wykonanie trasy do środka Placu Krakowskiego i powrót na obrzeża placu (Rysunek 3.3.2). Należy zauważyć, że każda z przebytych tras jest dobrze odwzorowana, jednak przesunięta o stały wektor. Jest to dobry znak, ponieważ testowany GPS prawidłowo wskazuje położenie modułu, a ewentualny błąd spowodowany jest niedokładnością mapy Google Earth.

3. Podsumowanie:

Możliwość udziału w projekcie: „Dobór i badania modułów nawigacji satelitarnej z przeznaczeniem do bezzałogowych platform latających" była dla nas ekscytującym przeżyciem i cennym doświadczeniem. Po raz pierwszy mieliśmy styczność z elementami elektronicznymi w tak bezpośrednim ujęciu. Dodatkowo, mogliśmy doświadczyć zalet jak i trudności pracy w zespole. Fakt, iż z projektu trzeba się rozliczyć, był dla nas tylko dodatkową mobilizacją.
Zrealizowany przez nas projekt jest problemem krytycznym w odniesieniu do tematu bezzałogowych obiektów latających, którymi zajmuje się Międzywydziałowe Koło Naukowe High Flyers.
Stworzony przez nas układ umożliwia wyświetlanie informacji o aktualnym położeniu (w odniesieniu geograficznym) obiektu, o prędkości jego poruszania się i azymucie. Dodatkowo, jesteśmy w stanie nanieść pobrane pomiary w celu wizualizacji przebytej trasy na mapie.
Biblioteki programowe stworzone w oparciu o język C# stanowią gotowy do implementacji moduł dla bezzałogowego obiektu latającego (o ile jest on wyposażony w silną jednostkę sterowania, np. system wbudowany). Natomiast funkcje i procedury stworzone na mikrokontroler AVR w języku C pozwalają na zaimplementowanie modułu GPS nawet w miniaturowym obiekcie, w którym pobór prądu odgrywa istotną rolę.
Podczas trwania projektu stworzyliśmy również funkcje zawierające obsługę peryferii mikrokontrolera AVR, między innymi takich jak: port szeregowy USART, przyciski funkcyjne, czy wyświetlacz LCD. Funkcje te możemy wykorzystać w kolejnych projektach.
Reasumując, ukończony przez nas projekt zakończył się stworzeniem bazy oprogramowania, sprzętu oraz wiedzy, którą będziemy mogli wykorzystać w przyszłości, w której chcielibyśmy rozpocząć pracę nad płytką drukowaną dla GPS.