**UNIWERSYTET GDAŃSKI**
**Wydział Matematyki, Fizyki i Informatyki**

**Daniel Sienkiewicz**
nr albumu: 206358

# Projekt komputera samochodowego bazujący na systemie mikrokomputera Intel Galileo

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
**dr inż. Janusz Młodzianowski**

Gdańsk 2015

## Streszczenie

Celem pracy jest stworzenie komputera pokładowego do samochodu, w którego skład wchodzi:

1. Mikrokomputer Intel Galileo Gen 1,
2. Ekran dotykowy FTDI VM800,
3. Oprogramowanie,
4. Kamerka cofania.



## Słowa kluczowe

Intel Galileo, $I^2C$, SPI, C, Arduino, GPIO, FTDI Chip, VM800

# Spis treści

ROZDZIAŁ 1

# Wprowadzenie

## 1.1. Cele

Celem pracy jest budowa oraz oprogramowanie komputera pokładowego do samochodu. Komputer powinien móc wczytać z czujników temperaturę panującą w silniku, na zewnątrz oraz w środku samochodu. Ponadto powinien on móc zapisać aktualną pozycję *GPS* na karcie pamięci microSD oraz umożliwić korzystanie z kamerki cofania lub inteligentnego lusterka wstecznego. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie ekranu dotykowego *FTDI Chip VM800*.

## 1.2. Założenia

Do wykonania komputera wykorzystano: *Intel Galileo* wraz z niezainstalowanym oprogramowaniem Linux YOCTO, Arduino IDE, lokalizator GPS służący do podawania aktualnej pozycji dzięki której obliczana zostaje droga przebyta przez samochód, kamerka internetowa służąca jako czujnik cofania oraz inteligentne lusterko wsteczne oraz symulator samochodu. Aktualnie komputer nie będzie zamontowany do fizycznego samochodu więc do tych celów zbudowany został symulator składający się z podstawowych czujników takich jak: guziki służące za czujnik zapięcia pasów/zamknięcia drzwi, potencjometry służące za czujniki temperatury oraz I/O expander PCF8574N pozwalający na komunikację z Intel Galileo. Na komputerze nie będzie wyświetlana aktualna prędkość ani przebieg ponieważ nawet w najnowszych samochodach nie jest to dostępna opcja. Dane te są dostępne na zegarach samochodowych więc nie ma potrzeby powtarzania tej informacji.

## 1.3. Plan pracy

Pierwszy rozdział opisuje podstawowe cele oraz założenia projektu.

Druga część pracy przedstawia architekturę projektu wraz z jego opisem funkcjonalnym. Opisuję również mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem.

W następnym rozdziale przedstawiony zostaje pomysł implementacji oraz proces tworzenia niezbędnej do obsługi symulatora samochodu biblioteki pozwalającej na komunikację poprzez I/O Expander PCF8574N z Intel Galielo. Zostaje tutaj również opisany proces tworzenia oprogramowania ekranu dotykowego FTDI Chip VM800.

Ostatnia część pracy przedstawia pomysły możliwych rozszerzeń projektu o dodatkowe moduły oraz funkcjonalności w zależności od potrzeb użytkownika.

ROZDZIAŁ 2

# Architektura

### 2.0.1. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem

#### 2.0.1.1. Porty

Porty są jednym z najbardziej podstawowych interfejsów. Najczęściej dzieli się je na porty:

1. Cyfrowe
2. Analogowe

Porty cyfrowe charakteryzują się możliwością przyjęcia lub wysłania sygnału binarnego (1 - jest sygnał, 0 - sygnału nie ma). Z kolei porty analogowe mogą przesyłać sygnały nawet 10 bitowe. Każdy z portów może działać w jednym z dwóch trybów: wejścia - oczekiwać na przyjęcie danych od urządzenia zewnętrznego oraz wyjścia - wysyłać dane do urządzenia zewnętrznego.

W środowisku Arduino aby obsłużyć port analogowy wystarczy:

**Listing 2.1**. Obsługa portu analogowego w środowisku Arduino

```
int val = 0;
int analogPin = A1;
pinMode(analogPin, OUTPUT);
val = analogRead(analogPin);
pinMode(analogPin, INPUT);
analogWrite(ledPin, val);
```

oraz odpowiednio dla portu cyfrowego:

**Listing 2.2**. Obsługa portu cyfrowego w środowisku Arduino

```
int val = 0;
int digitalPin = 1;
pinMode(digitalgPin, OUTPUT);
val = digitalRead(analogPin);
pinMode(digitalPin, INPUT);
digitalWrite(digitalPin, HIGH);
```

#### 2.0.1.2. Przerwania

Przerwania są to bezpośrednie funkcje systemu lub sprzętu ułatwiające komunikację ze światem zewnętrznym. Część z nich jest zarezerwowana przez system lecz część z nich jest wolna do wykorzystania dla programisty. Przerwania możemy podzielić na trzy podstawowe rodzaje:

1. Programowe
2. Sprzętowe
   (a) Maskowalne (NMI)
   (b) Niemaskowalne (INTR)
3. Wyjątek

Przerwania programowe wywołuje się za pomocą komendy INT XX gdzie XX oznacza numer przerwania zadeklarowanego w tablicy wektorów przerwań, która jest tworzona przy każdorazowym starcie systemu. Przerwanie to może przyjąć wartości do 255 i są one zarezerwowane przez procesor oraz użytkownika.

Przerwanie sprzętowe jest to rodzaj przerwań wywoływanych przez urządzenia wejścia/wyjścia lub zgłaszane przez procesor. Zostają one wywołane niezależnie w określonych przypadkach. Przerwania te dzielimy na maskowalne oraz niemaskowalne. Główna różnica między nimi polega na możliwości

zablokowania przerwań maskowalnych podczas gdy przerwania niemaskowalne muszą zostać obsłużone. Przykładem przerwania niemaskowalnego INT2 czyli popularny blue screen of death.

Ostatnim rodzajem przerwań są wyjątki. Wywoływane są podczas napotkania przez procesor błędów oraz niepowodzeń. Arduino oczywiście obsługuje przerwania. Ich obsłga jest bardzo prosta: W środowisku Arduino aby obsłużyć port analogowy wystarczy:

**Listing 2.3**. Obsługa przerwań sprzętowych w środowisku Arduino

```
attachInterrupt(pinInt, funcName, mode);
```

gdzie pinInt jest to pin na którym Arduino będzie nasłuchiwało na przerwanie, funcName jest to nazwa funkcji który zostanie wykonana gdy przerwanie zostanie zgłoszone, mode - jest to określenie kiedy sygnał może być uznany za przerwanie.

#### 2.0.1.3. Odpytywanie w pętli

Jednym z najprostszych metod pozyskania danych z mikro kontrolera jest jego odpytywanie w nieskończonej pętli.Jest to najmniej efektywny sposób ponieważ cały czas zajmuje niepotrzebnie zasoby sprzętu niepotrzebnymi zapytaniami.

**Listing 2.4**. Odpytywanie w nieskończonej pętli w środowisku Arduino

```
void loop() {
        funcName();
        delay(1000);
}
```

#### 2.0.1.4. Timer

**Listing 2.5**. Użycie timer w środowisku Arduino

```
#include <TimerOne.h>
Timer1.initialize(500000); // ustawienie ś długoci timera
Timer1.attachInterrupt(funcName, 500000);
```

#### 2.0.1.5. Protokół komunikacyjny

TO DO

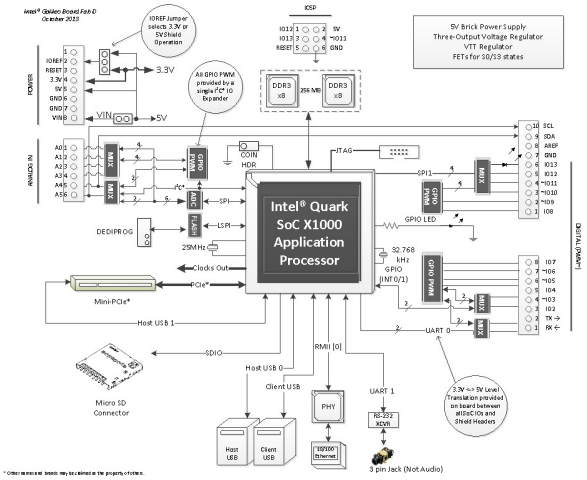
#### 2.0.1.6. Złączki oraz kable

TO DO

### 2.0.2. Intel Galileo

Intel Galileo jest to mikro kontroler oparty na 32-bitowym procesorze Intel® Quark SoC X1000 i taktowaniu 400MHz. Został on wyposażony w 14 pinów cyfrowych (w tym 6 pinów mogących pełnić funkcję PWM) oraz 6 pinów cyfrowych.Każdy z tych pinów jest w stanie operować napięciem max 5V. Bardzo dużym atutem Galileo jest wbudowana karta sieciowa, port RS-232 oraz port USB oraz slot karty microSD. Galileo może być używane w dwóch trybach - trybie w pełni kompatybilnym z Arduino oraz w trybie z zainstalowanym systemem operacyjnym (np. Linux).

ŹRÓDŁO: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo




**Rysunek 2.1**. Schemat logiczny układu Intel Galileo

Źródło: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo

ROZDZIAŁ 3

# Implementacja

## 3.1. Protokół komunikacyjny $I^2C$

$I^2C$ jest szeregowym interfejsem służącym do przesyłania danych między urządzeniami elektrycznymi.

Podstawową cechą $I^2C$ jest wykorzystywanie dwóch linii służących do komunikacji: linia *SDA (Serial Data Line)* oraz linia *SCL(Serial Clock Time)*. Każdą transmisję danych należy rozpocząć sygnałem *START* oraz zakończyć sygnałem *STOP*. Dane wysyłane są od najstarszego do najmłodszego bitu oraz otrzymanie każdego z nich musi być potwierdzone przez odbiornik. Należy również pamiętać aby każdą komunikację z urządzeniem rozpocząć i zakończyć ustawiając linie *SDA* oraz *SCL* w stan nieaktywny (HIGH). Podstawowe zalety protokołu:

1. Połączenia składają się tylko z dwóch linii co znacznie ogranicza liczbę kabli wychodzących z urządzenia

2. Duża dostępność sprzętu w sklepach

3. Transmisja jest odporna na zakłócenia zewnętrzne

4. Bez większych problemów można dodawać oraz odejmować układy korzystające z magistrali

Odbieranie danych rozpoczyna się wysłaniem sygnału START, a następnie zaadresowaniu urządzenia. Kolejnym krokiem jest ustalenie tryby (w tym wypadku read) oraz odczytanie potwierdzenia. Po wykonaniu tych czynności można rozpocząć odbierania danych, które należy zakończyć wysłaniem sygnału *STOP*.

Wygenerowanie sygnału *START* polega na ustawieniu linii *SDA* oraz *SCL* w stan niski (LOW), a wygenerowanie sygnału *STOP* polega na ustawieniu linii *SDA* oraz *SCL* w stan wysoki (HIGH).

Adresowanie urządzenia odbywa się poprzez wysłanie wysłanie pojedynczych bitów adresu (pamiętając o kolejności MSB->LCB) oraz wygenerowanie impulsu zegara.

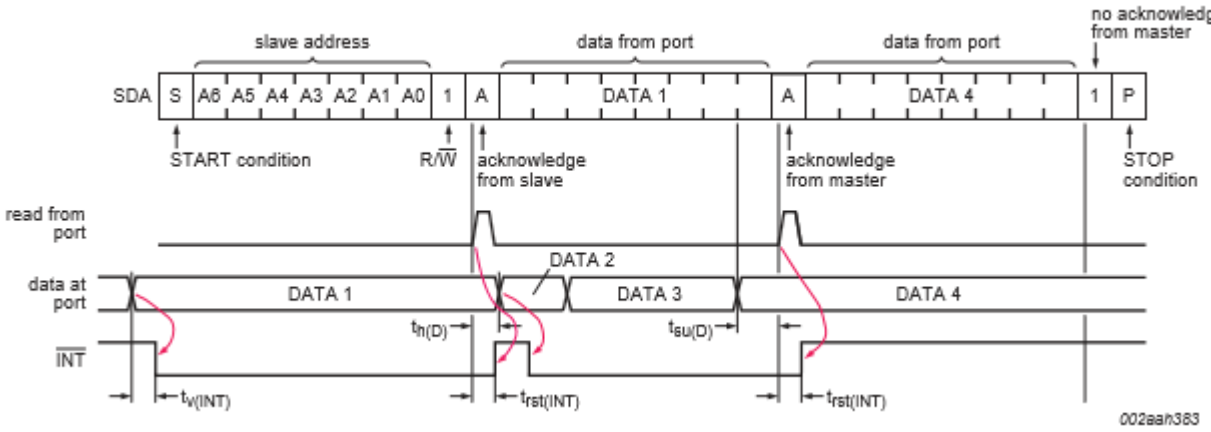
**Listing 3.1**. Adresowanie urządzenia $I^2C$ na przykładzie PCF8574N

```
for(m = 0x80; m; m >>= 1){
    if(adres & m)
      digitalWrite(sda, HIGH);
    else
      digitalWrite(sda, LOW);

  digitalWrite(scl, HIGH);
  digitalWrite(scl, LOW);
}
```

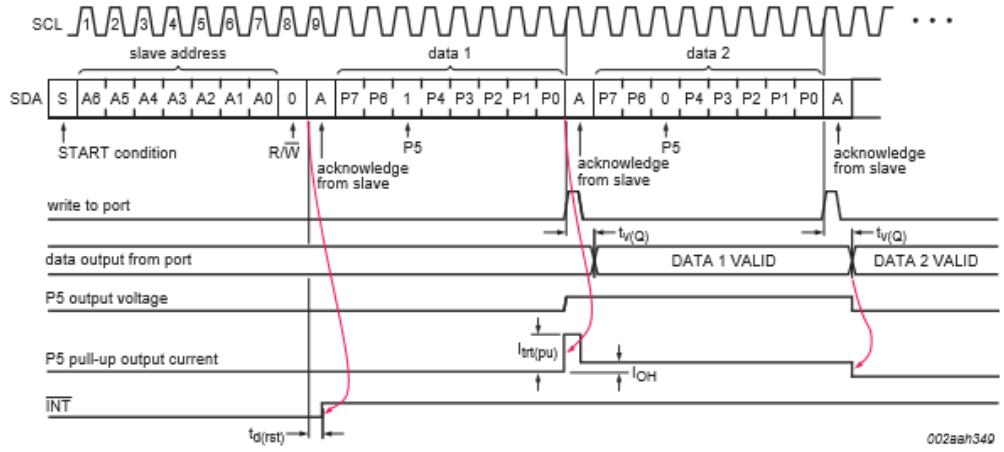
Po otrzymaniu potwierdzenia na linii *SDA* można zacząć czytać dane przesyłane z urządzenia.

Podobnie jak odbieranie danych, wysyłanie danych należy rozpocząć od wysłania sygnału *START* wraz z adresem urządzenia oraz trybem (write). Po otrzymaniu potwierdzenia można rozpocząć wysyłanie danych w odpowiednich dla urządzenia paczkach x-bitowych. Po wysłaniu każdej z nich otrzymamy potwierdzenie. Na zakończenia transmisji należy wysłać sygnał *STOP*.




**Rysunek 3.1**. Przykładowy schemat odbierania danych poprzez $I^2C$ na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N




**Rysunek 3.2**. Przykładowy schemat wysyłania danych poprzez $I^2C$ na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

### 3.1.1. Problemy z bibliotekami

### 3.1.2. Własna implementacja $I^2C$

### 3.1.3. Schemat blokowy programu

### 3.1.4. Własna biblioteka do komunikacji poprzez $I^2C$ dla Intel Galileo

## 3.2. Założenia funkcjonalne projektu

Najważniejszym założeniem funkcjonalnym była komunikacja z zestawem czujników, które mogą być zamontowane w samochodzie. czytanie z czujników, pisanie do ekranu, czytanie z ekranu włączanie i wyłączanie systemu

## 3.3. Integracja z samochodem

- podpięcie pod auto - włączanie i wyłączanie systemu - można brutalnie wyłączyć

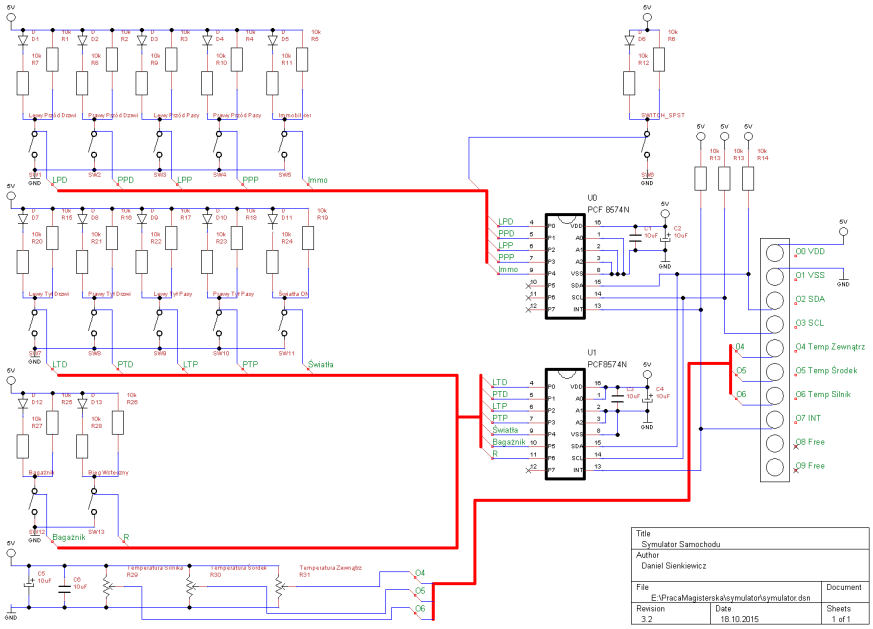
### 3.3.1. Symulator samochodu

## 3.4. VM800

na poczatku emulacja na PC potem przepisanie na niski poziom ostatecznie podpiecie do Galielo (poszukac czy juz jest?) programers manual reference vm800 ftdi POSZUKAĆ!!!!

## 3.5. Dalsze kroki oraz propozycje

schemat blokowy z BAJERAMI i wybrane to co zrobię




**Rysunek 3.3**. Schemat symulatora samochodu

Źródło: Opracowanie własne

# Zakończenie

TO DO

DODATEK A

# Karty Katalogowe

Katalog *datasheets* zawiera karty katalogowe użytych podzespołów

DODATEK B

# Porównanie dostępnych na rynku mikro kontrolerów

| | Intel Galileo | Raspberry Pi (Model B) | Arduino Uno |
|---|---|---|---|
| Wymiary | 10cm x 7cm | 85.60mm x 56mm x 21mm | 5.59cm x 16.5cm |
| Procesor | Intel Quark X1000 | Broadcom BCM2835 | ATmega328 |
| Taktowanie | 400MHz | 700MHziv | 16 MHz |
| Cache | 16 KB | 32KB L1 cache, 128KB L2 cache | - |
| RAM | 512 SRAm | 512 SRAM | 2 kB |
| Analog I/O | 6 | 17 | 6 |
| Digital I/O | 14 | 8 | 14 |
| PWM | 6 | 1 | 6 |

**Tabela B.1**. Specyfikacja dostępnych na rynku mikro kontrolerów

Źródło: http://eu.mouser.com/applications/open-source-hardware-galileo-pi/http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html

**DODATEK C**

# Programy

Katalogi *Galileo, PCF8574N* zawierają kod źródłowy oprogramowania stworzonego na potrzeby pracy.
Katalog *Galileo* zawiera oprogramowanie mikrokomputera Intel[1] Galileo.
Katalog *PCF8574N* zawiera oprogramowanie I/O Expander PCF8574N.

# Bibliografia

[1] Agjffgjgdjfgflbert ffdddffEinstein. hkljklbkln. *Annalen der Physik*, 322(10):891–921, 1905.

# Spis tabel

# Spis rysunków

# Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

| | |
|---|---|
| ..... | ..... |
| data | podpis |