

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. Kamera cofania.

Słowa kluczowe

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

Spis treści

1. Wprowadzenie	5
1.1. Cele	5
1.2. Założenia	5
1.3. Plan pracy	6
2. Architektura	7
2.0.1. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem	7
3. Implementacja	11
3.0.1. Intel Galileo	11
3.1. Protokół komunikacyjny I^2C	11
3.1.1. Użycie protokołu I^2C na przykładzie I/O Expander PCF8574N	13
3.1.2. Problemy z bibliotekami	15
3.1.3. Własna implementacja I^2C	15
3.1.4. Własna biblioteka do komunikacji poprzez I^2C dla Intel Galileo	15
3.2. Protokół komunikacyjny SPI	15
3.2.1. Komunikacja poprzez protokół SPI	16
3.2.2. VM800	17
3.3. Założenia funkcjonalne projektu	18
3.3.1. Symulator samochodu	18
3.4. Dalsze kroki oraz propozycje	18
Zakończenie	20
A. Karty Katalogowe	21
B. Porównanie dostępnych na rynku mikro kontrolerów	22

C. Programy	23
Bibliografia	24
Spis tabel	25
Spis rysunków	26
Oświadczenie	27

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 kamery cofania lub inteligentnego lusterka wstecznego. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie ekranu dotykowego *FTDI EVE 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, kamera 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 Galileo*. Zostaje tutaj również opisany proces tworzenia oprogramowania ekranu dotykowego FTDI EVE 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 - wysłać 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(digitalPin , 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 obsługa 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. Przykładowe użycie timer w środowisku Arduino

```
#include <TimerOne.h>
Timer1.initialize(500000);
Timer1.attachInterrupt(funcName, 500000);
```

ROZDZIAŁ 3

Implementacja

3.0.1. 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).

3.1. Protokół komunikacyjny I^2C

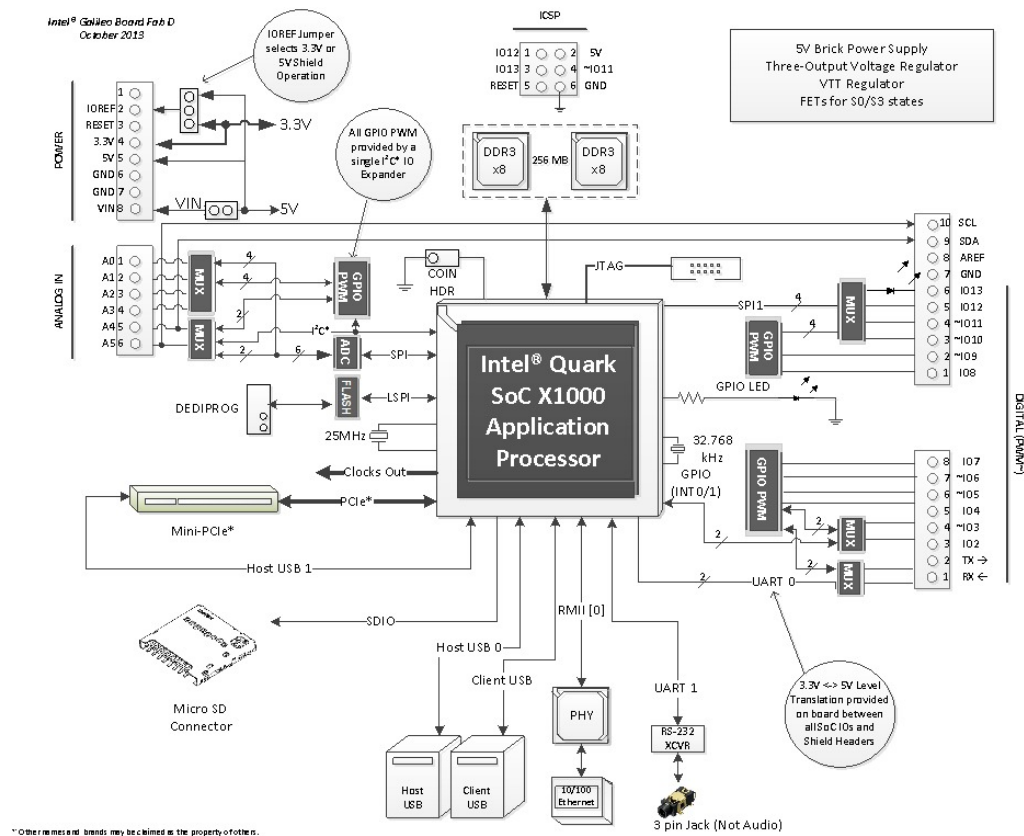
I^2C jest szeregowym interfejsem służącym do przesyłania danych między urządzeniami elektrycznymi.

Podstawową cechą I^2C jest wykorzystywanie dwóch linii służących do komunikacji: linia SDA ¹ oraz linia SCL ². 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

¹Serial Data Line

²Serial Clock Time

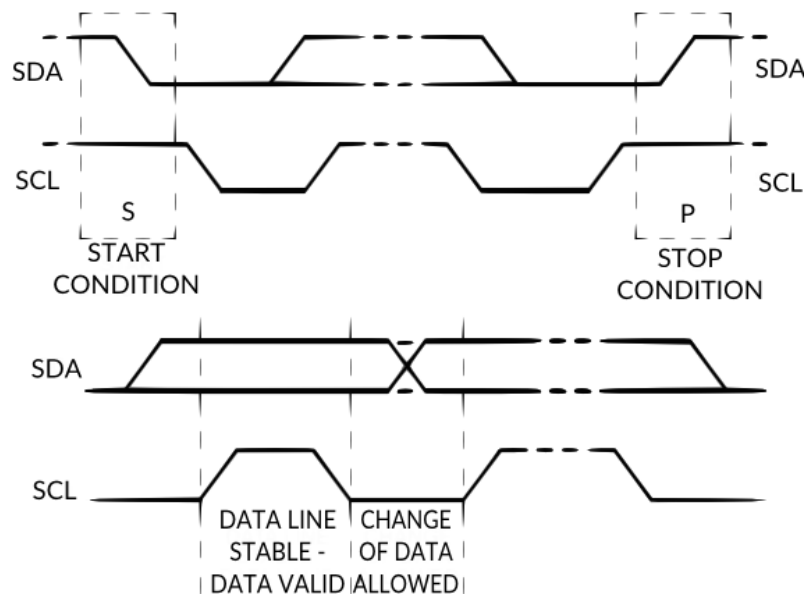


Rysunek 3.1. Schemat logiczny układu Intel Galileo

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

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*.



Rysunek 3.2. Przebieg czasowy protokołu I^2C

Źródło: <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>

3.1.1. Użycie protokołu I^2C na przykładzie I/O Expander PCF8574N

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)
```

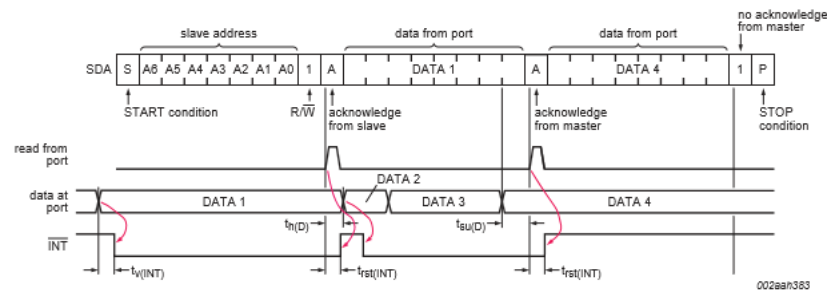
```

        digitalWrite(sda, HIGH);
    else
        digitalWrite(sda, LOW);

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

```

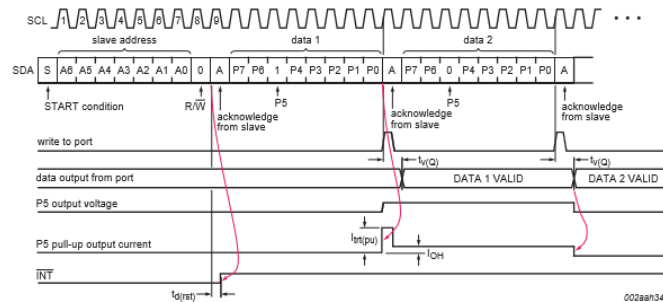
Po otrzymaniu potwierdzenia na linii *SDA* można zacząć czytać dane przesyłane z urządzenia.



Rysunek 3.3. Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

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 3.4. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

3.1.2. Problemy z bibliotekami

3.1.3. Własna implementacja I^2C

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

3.2. Protokół komunikacyjny SPI

Kolejnym przykładem interfejsu szeregowego jest protokół SPI³. Składa się on z trzech podstawowych linii - dwóch służących do przesyłania danych w przeciwnych kierunkach oraz jednej z sygnałem taktującym synchronizującym transfer danych.

Linia MISO⁴ jest linią wejścia danych dla urządzenia nadrzędnego (master), a wyjściem dla urządzenia podrzędnego (slave), linia MOSI⁵ jest wyjściem dla urządzenia master, a wejściem dla slave. Linia SCK⁶ jest wejściem taktującym. Sygnał taktujący jest zawsze generowany przez układ master.

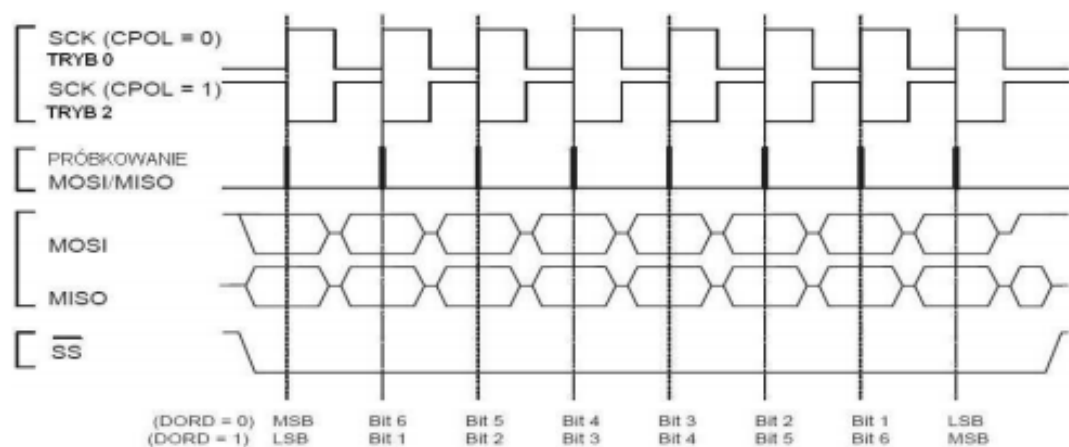
³Serial Peripheral Interface

⁴Master In Slave Out

⁵Master Out Slave In

⁶Serial Clock

Transmisja danych na obydwu liniach jest zawsze dwukierunkowa i odbywa się jednocześnie - nadanie danych na linii MISO wiąże się z nadaniem danych na linii MOSI. Nie zawsze jednak nadane dane niosą ze sobą informację - najczęściej nadawane informacje płyną w jedną stronę podczas gdy w tym samym czasie wysyłane zostają puste dane. Parametry CPOL⁷ oraz CPHA⁸ określają polaryzację oraz fazę sygnału zegarowego.



Rysunek 3.5. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0

Źródło: <http://castor.am.gdynia.pl/~dorra>

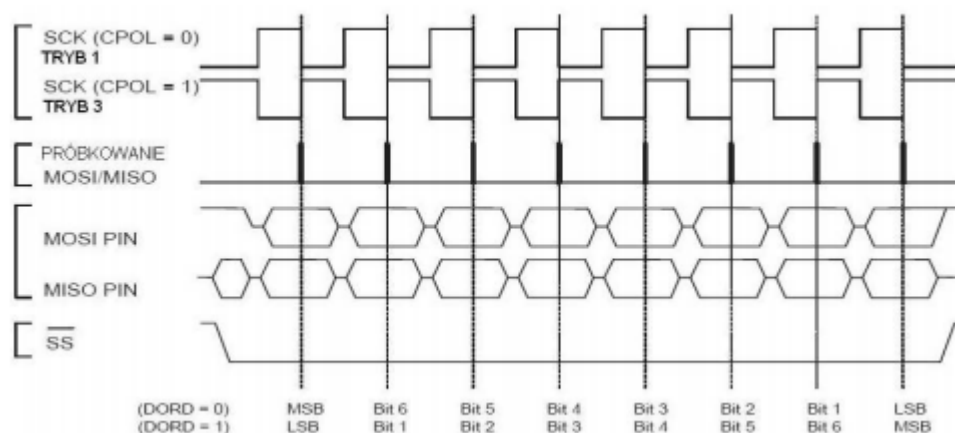
3.2.1. Komunikacja poprzez protokół SPI

Komunikacja zawsze przebiega dwustronnie (duplex⁹). Zmienia taktu zegara z niskiego na wysoki daje możliwość odczytania danych z linii MISO i jednocześnie wysłania danych na linię MOSI. W przypadku zmiany taktu zegara z wysokiego na niski dane zostają wysłane poprzez linię MISO i jednocześnie odczytane z linii MOSI.

⁷Clock Polarity

⁸Clock Phase

⁹Nadawanie i odbieranie informacji odbywa się w obu kierunkach



Rysunek 3.6. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=1

Źródło: <http://castor.am.gdynia.pl/~dorra>

	Zbocze pierwsze	Zbocze ostatnie	Tryb SPI
CPOL = 0, CPHA = 0	Pobieranie (narastanie)	Pobieranie (opadanie)	0
CPOL = 0, CPHA = 1	Ustawianie (narastanie)	Ustawianie (opadanie)	1
CPOL = 1, CPHA = 0	Pobieranie (opadanie)	Pobieranie (narastanie)	2
CPOL = 1, CPHA = 1	Ustawianie (opadanie)	Ustawianie (narastanie)	3

Tabela 3.1. Tryby reakcji na fazę i biegunowość sygnału zegarowego

Źródło: <http://atmega32.republika.pl/22.htm>

3.2.2. VM800

FDTI EVE VM800 jest to wyświetlacz dotykowy wraz z wbudowanym kontrolerem audio. Podstawowe cechy urządzenia:

1. Pojedynczy układ scalony dla wyświetlacza oraz kontrolera Audio
2. Rozdzielczość ekranu do 512 x 512 pikseli

3. 262 tys. kolorów
4. Możliwość wygładzania krawędzi
5. Możliwość komunikacji poprzez użycie interfejsu I^2C lub SPI
6. Wbudowane widgety dostępne dla użytkownika
7. Zakres pracy wyświetlacza: $-40^{\circ}C$ do $85^{\circ}C$

3.3. 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

3.3.1. Symulator samochodu

3.4. Dalsze kroki oraz propozycje

schemat blokowy z BAJERAMI i wybrane to co zrobić

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:](http://eu.mouser.com/applications/open-source-hardware-galileo-pi/)

[//eu.mouser.com/applications/open-source-hardware-galileo-pi/](http://eu.mouser.com/applications/open-source-hardware-galileo-pi/)

Źródło: [http:](http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html)

[//botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html](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] Agjffgjdjfgflbert fdddfEinstein. hkljklbkln. *Annalen der Physik*, 322(10):891–921, 1905.

Spis tabel

3.1. Tryby reakcji na fazę i biegunowość sygnału zegarowego	17
B.1. Specyfikacja dostępnych na rynku mikro kontrolerów	22

Spis rysunków

3.1. Schemat logiczny układu Intel Galileo	12
3.2. Przebieg czasowy protokołu I^2C	13
3.3. Przykładowy schemat odbierania danych poprzez I^2C na przy- kładzie PCF8574N	14
3.4. Przykładowy schemat wysyłania danych poprzez I^2C na przy- kładzie PCF8574N	15
3.5. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0	16
3.6. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=1	17
3.7. Schemat elektryczny symulatora samochodu	19

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