

**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 2016

## **Streszczenie**

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

1. Mikrokomputer Intel Galileo Gen 1,
2. Wyświetlacz TFT FTDI EVE VM800B z panelem dotykowym,
3. Zestaw czujników symulujących odpowiedni dla rzeczywistego samochodu stan w szczególności:
  - (a) Informacje na temat temperatury w samochodzie, na zewnątrz oraz w silniku
  - (b) Informacje na temat otwarcia/zamknięcia drzwi
  - (c) Informacje na temat odpięcia/zapięcia pasów
4. Oprogramowanie

Dodatkowym celem było praktyczne sprawdzenie możliwości oprogramowania i funkcjonalności systemu Intel Galileo.

## **Słowa kluczowe**

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

# Spis treści

<b>1. Wprowadzenie</b>	5
1.1. Cele	5
1.2. Założenia	5
1.3. Plan pracy	6
<b>2. Architektura</b>	7
2.0.1. Arduino	7
2.0.2. Intel Galileo	8
2.0.3. FTDI EVE VM800B	10
2.0.4. Symulator samochodu	12
<b>3. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem</b>	15
3.0.1. Porty wejścia/wyjścia	15
3.0.2. Odpytywanie w pętli	15
3.0.3. Przerwania	16
<b>4. Programowanie Intel Galileo z użyciem różnych środowisk</b>	18
4.1. Programowanie w środowisku Intel Arduino studio	18
4.2. Komunikacja z urządzeniami poprzez mechanizmy systemu operacyjnego Linux YOCTO	22
<b>5. Implementacja</b>	24
5.1. Protokół komunikacyjny $I^2C$	24
5.1.1. I/O Expander PCF8574N	28
5.2. Protokół komunikacyjny SPI	29
5.2.1. Komunikacja z ekranem FTDI EVE VM800b poprzez protokół SPI	30

<b>6. Działanie komputera pokładowego . . . . .</b>	33
6.1. Założenia funkcjonalne projektu . . . . .	33
6.2. Opis budowy i działania . . . . .	35
6.3. Wnioski oraz własne doświadczenia . . . . .	43
<b>Zakończenie . . . . .</b>	45
<b>A. Karty Katalogowe . . . . .</b>	47
<b>B. Porównanie dostępnych na rynku mikro kontrolerów . . . . .</b>	48
<b>C. Mapowanie portów Intel Galileo na pliki w systemie Linux . . . . .</b>	49
<b>D. Programy oraz dokumentacja . . . . .</b>	50
<b>Bibliografia . . . . .</b>	51
<b>Spis tabel . . . . .</b>	53
<b>Spis rysunków . . . . .</b>	54
<b>Oświadczenie . . . . .</b>	55

## **ROZDZIAŁ 1**

# **Wprowadzenie**

### **1.1. Cele**

Celem pracy była konstrukcja oraz oprogramowanie systemu komputera pokładowego do samochodu. Komputer ma wczytywać temperaturę panującą w silniku, na zewnątrz, w środku oraz aktualne stany zapięcia pasów i zamknięcia drzwi. Jednym z założeń tworzenia systemu jest konstrukcja modułowa umożliwiająca późniejszą rozbudowę funkcjonalności o dodatkowe funkcje na przykład funkcję rejestrującą pozycję *GPS*. Dodatkową funkcjonalnością jest możliwość zapisania danych obrazujących aktualny stan samochodu na karcie pamięci *microSD*. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie wyświetlacza TFT *FTDI EVE VM800B* z panelem dotykowym.

### **1.2. Założenia**

Do wykonania pracy zostały przyjęte następujące założenia:

1. Użycie mikrokomputera *Intel Galileo Gen 1* lub *Intel Galileo Gen 2* jako głównego silnika dla całego komputera wraz z zainstalowanym systemem operacyjnym *Linux YOCTO*
2. Użycie wyświetlacza TFT *FTDI EVE VM800B* z panelem dotykowym jako interfejsu komunikacyjnego komputera z użytkownikiem,
3. Kompilacja oprogramowania przy użyciu systemu Intel Arduino studio oraz natywnego systemu dla Galileo - Linux YOCTO - kompilator *GCC*
4. Symulacja funkcjonalności rzeczywistych czujników samochodu za pomocą symulatora składającego się z zestawu przełączników symulujących

cych stan zapięcia pasów/zamknięcia drzwi oraz potencjometrów służących do symulacji analogowych czujników temperatury

Parametry takie jak prędkość oraz przebieg nie będą rejestrowane ponieważ są one standardowo dostępne na zegarach samochodowych więc nie ma potrzeby powtarzania tej informacji.

### **1.3. Plan pracy**

Praca została podzielona na dwie części. Pierwsza z nich jest częścią teoretyczną opisującą działanie użytych w projekcie komponentów oraz sposobów komunikacji pomiędzy nimi.

W rozdziale *Architektura* zostały opisane komponenty użyte do stworzenia pracy: Intel Galileo, FTDI EVE VM800B oraz symulator samochodu. Następnie opisane zostały mechanizmy komunikacji systemu z otoczeniem takie jak: Porty, Odpytywanie w pętli, Przerwania.

Część druga poświęcona jest praktycznej stronie projektu. W rozdziale *Implementacja* wyjaśnione zostało jak działają użyte w projekcie protokoły komunikacyjne wraz z opisem własnych wersji bibliotek napisanych na potrzeby pracy oraz ich wykorzystania w projekcie. Na koniec opisane zostało działanie stworzonego komputera pokładowego wraz z jego wszystkimi możliwościami i propozycjami dalszej rozbudowy.

## ROZDZIAŁ 2

# Architektura

### 2.0.1. Arduino

*Arduino* jest to platforma OPEN-SOURCE<sup>1</sup> bazująca na łatwym do użycia oprogramowaniu oraz urządzeń. Na płytce Arduino w zależności od wersji programista ma do dyspozycji od 14 pinów cyfrowych i 6 analogowych, port Ethernet oraz USB/microUSB. Nazwa Arduino obowiązuje tylko w USA. W pozostałych krajach ten sam sprzęt jest dostępny pod nazwą Genuino.



Rysunek 2.1. Arduino Uno

Źródło: <https://www.arduino.cc/en/Main/ArduinoBoardUno>[1]

Największą zaletą Arduino jest charakterystyczne i zawsze takie same rozmieszczenie dostępnych pinów. Z tego powodu wielu producentów w swoich produktach uwzględnia to rozmieszczenie przez co podłączenie zewnętrznych komponentów jest bardzo proste i wygodne.

---

<sup>1</sup>Licencja oprogramowania, w myśl której cały użyty kod źródłowy jest w pełni dostępny dla programisty

Najczęściej spotykane wersje Arduino to:

1. Arduino Uno - najbardziej podstawowa wersja
2. Arduino Leonardo
3. Arduino Mega - wersja z dużo większą ilością wejść GPIO
4. Arduino Pro

### 2.0.2. Intel Galileo

Zestaw Intel Galileo jest to mikrokomputer oparty na 32-bitowym procesorze Intel® Quark SoC X1000 i taktowaniu 400MHz. Posiada on standardowy interfejs Arduino składający się z: 14 pinów cyfrowych (w tym 6 pinów mogących pełnić funkcję *PWM*<sup>2</sup>) oraz 6 pinów analogowych. Intel Galileo jest zgodny z standardem TTL<sup>3</sup> wartość logiczna 1 równa jest napięciu 5V, a wartość 0 odpowiada napięciu 0V. W Arduino odpowiednikiem tego są wartości *HIGH* oraz *LOW*.

Intel Galileo zostało wyposażone w:

1. Wbudowaną kartę sieciową IEEE 802.3,
2. Port *RS-232* oraz *USB*,
3. Złącze mini PCI Express,
4. Przetwornik analogowo-cyfrowy (10-bitowy),
5. Interfejs *SPI* oraz *I<sup>2</sup>C*,
6. Zegar czasu rzeczywistego (RTC<sup>4</sup>)
7. UART<sup>5</sup>,

---

<sup>2</sup>ang. Pulse-Width Modulation - technika pozyskiwania wyników analogowych poprzez użycie wyjść cyfrowych

<sup>3</sup>ang. Transistor-transistor logic

<sup>4</sup>ang. Real-Time Clock

<sup>5</sup>ang. Universal Asynchronous Receiver and Transmitter - układ używany do asynchronicznego przekazywania informacji poprzez port szeregowy

8. Slot karty *microSD*.[2]



**Rysunek 2.2.** Galileo Gen 1 Board

Źródło: <http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g1-datasheet.html>[3]

Do komunikacji z Intel Galileo programista ma do dyspozycji port RS-232, USB (działające w trybie host oraz client) oraz wyjście Ethernet. Intel Galileo jest zasilane napięciem 5V 2.0A, które może zostać dostarczone poprzez zasilacz z zestawu lub poprzez podłączenie zasilania do portów PWR<sup>6</sup> oraz GND<sup>7</sup>. Standardowym środowiskiem programistycznym mikrokomputera jest Intel Arduino Studio. Programista pisząc w języku C/C++ i używając dostarczonych przez producenta Arduino funkcji do obsługi portów może wystawiać na nich stany logiczne. Następnie przesyła skompilowaną wersję oprogramowania poprzez kabel USB do urządzenia. Po przesłaniu program zostaje załadowany do pamięci urządzenia i uruchomiony.

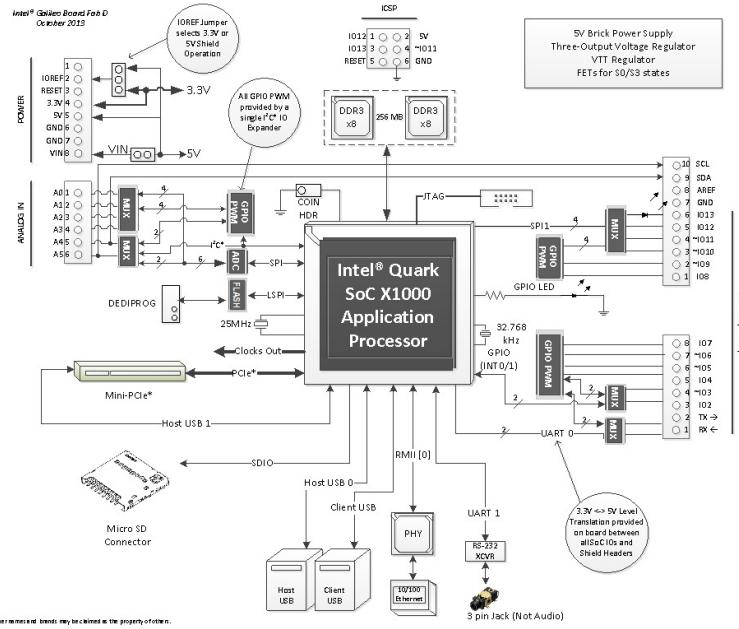
Standardowo w Intel Galileo znajduje się podstawowa wersja systemu mini Linux bazująca na dystrybucji *YOCTO*. Użytkownik może jednak użyć pełnego obrazu systemu uruchamiając go z karty pamięci. Komunikację z systemem operacyjnym można prowadzić w dowolnym języku programowania

---

<sup>6</sup>Port używany jako port zasilania (5V)

<sup>7</sup>Port używany jako masa - GROUND

(np. C, NodeJS, python) łącząc się poprzez dowolny port komunikacyjny na przykład: SSH<sup>8</sup> lub port RS-232.



**Rysunek 2.3.** Schemat logiczny układu Intel Galileo

Źródło: [https://www.arduino.cc/en/ArduinoCertified/IntelGalileo\[4\]](https://www.arduino.cc/en/ArduinoCertified/IntelGalileo[4])

### 2.0.3. FDTI EVE VM800B

Do komunikacji komputera z użytkownikiem został użyty wyświetlacz *TFT FDTI EVE VM800B* o rozdzielcości 800x600 wraz z panelem dotykowym oraz wbudowanym kontrolerem audio.

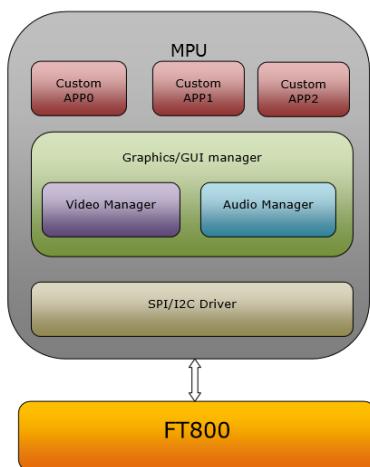
Podstawowe cechy urządzenia[5]:

1. Pojedynczy układ scalony dla wyświetlacza oraz kontrolera Audio
2. Ekran 3.5" TFT

---

<sup>8</sup>ang. secure shell - protokół komunikacyjny służący do połączenia się ze zdalnym komputerem będącym w sieci

3. Możliwość komunikacji poprzez użycie interfejsu  $I^2C$  lub SPI
4. Wbudowany system HMI<sup>9</sup> - stworzenie interfejsu pomiędzy użytkownikiem a systemem w bardzo prosty oraz wygodny sposób umożliwia system bazujący na widgetach.



**Rysunek 2.4.** Architektura FTDI EVE VM800B

Źródło: FT800 Programmers Guide

Rozpoczęcie pracy wyświetlacza polega na zainicjalizowaniu go poprzez wpisanie określonych przez specyfikację producenta wartości do określonych obszarów rejestrów określając w ten sposób np. rozdzielcość lub włączenie-/wyłącznie modułu odpowiedzialnego za dźwięk czy dotyk.

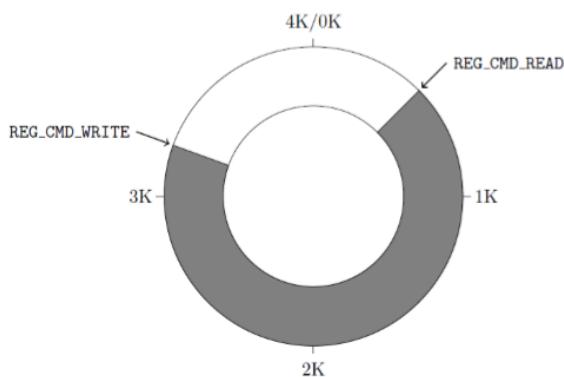
Kolejnym krokiem jest wywołanie zestawu funkcji odpowiedzialnych za rysowanie niezbędnych do działania systemu elementów np. guzików. Wszystkie wyświetlane elementy są na początku zapisywane w kołowym buforze pamięci. Następnie gdy cały ekran zostanie już przygotowany następuje wyświetlenie wszystkiego co zostało zapisane do bufora po czym zostaje on wyczyszczony. Należy pamiętać, że wielkość bufora, którą mamy do dyspozycji wynosi 4 Kb.

---

<sup>9</sup>ang. Human - Machine Interface

Procedura rysowania wygląda następująco:

1. Poczekaj aż wszystko co miało zostać wyświetcone, zostanie wyświetlone - opróżnij bufor
2. Określ co będzie rysowane - np. linia lub kropka i dodaj to do bufora i przesuń się o 4 bajty w buforze
3. Ustaw wszystkie potrzebne parametry - np. wielkość, kolor lub położenie i dodaj to do bufora, pamiętając aby za każdym razem przesunąć się o 4 bajty w buforze
4. Wyświetl wszystko do zostało dodane do bufora kołowego



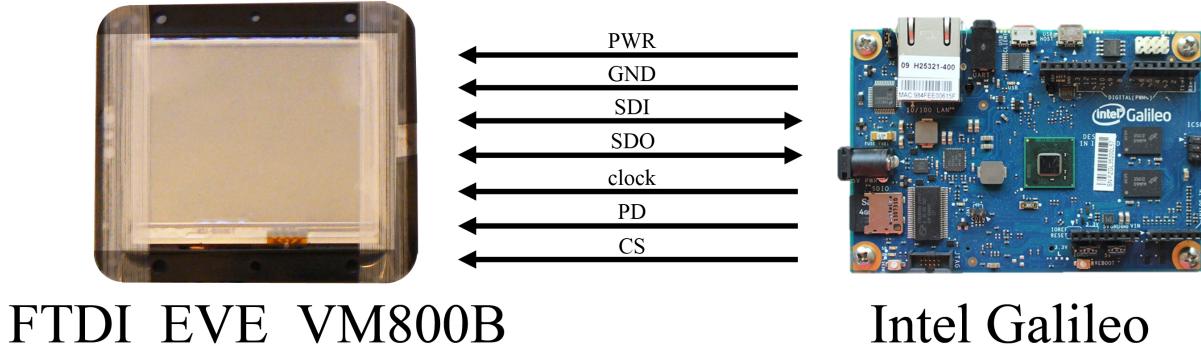
**Rysunek 2.5.** Bufor kołowy dostępny podczas programowania ekranu VM800

Źródło: FT800 Programmers Guide

Komunikacja Galileo z Ekranem odbywa się poprzez protokół komunikacyjny *SPI* (Protokół ten został opisany w części "Implementacja").

#### 2.0.4. Symulator samochodu

Do celów demonstracyjnych oraz implementacji oprogramowania komputer nie został zamontowany w fizycznym samochodzie. Zamiast tego został zbudowany symulator samochodu mający obrazować pełną pracę pojazdu.

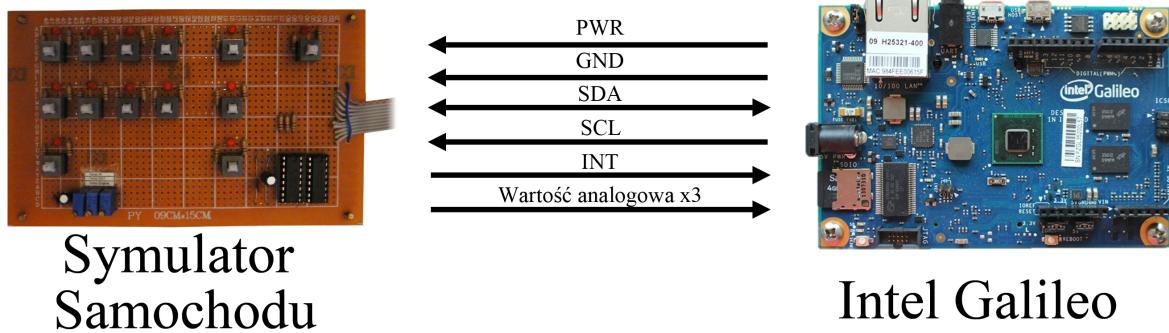


FTDI EVE VM800B

Intel Galileo

**Rysunek 2.6.** Schemat połączenia wyświetlacza FTDI EVE z kontrolerem za pomocą SPI

Źródło: Opracowanie własne



**Rysunek 2.7.** Schemat połączenia symulatora samochodu z kontrolerem za pomocą  $I^2C$

Źródło: Opracowanie własne

Symulator składa się z:

1. 11 przełączników bistabilnych symulujących stan drzwi samochodu (wraz z stanem bagażnika), pasy pasażerów, przełączenie biegu na wsteczny oraz włączenie świateł - wciśnięcie przełącznika dodatkowo obrazowane jest poprzez zaświecenie diody LED w symulatorze

2. 2 I/O Expandery PCF 8574N
3. 3 potencjometrów symulujących czujniki temperatury w samochodzie

Przełączniki wysyłają sygnały cyfrowe, a potencjometry sygnały analogowe. Ze względu na znaczną ilość sygnałów cyfrowych wychodzących z simulatorka komunikacja z Galileo odbywać się będzie poprzez *I/O Expander PCF 8574N* wykorzystujący protokół  $I^2C$  (Protokół ten został opisany w części "Implementacja").

## **ROZDZIAŁ 3**

# **Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem**

### **3.0.1. Porty wejścia/wyjścia**

Porty najczęściej dzieli się na:

1. Cyfrowe
2. Analogowe

Porty cyfrowe charakteryzują się możliwością przyjęcia lub wysłania sygnału binarnego (1 - jest sygnał, 0 - sygnału nie ma). W standardzie *TTL* wysłanie sygnału równego 1 jest równoznaczne z wysłaniem napięcia o wartości 5V oraz odpowiednio wysłanie 0 jest równoznaczne z wysłaniem napięcia równego 0V. Każdy z portów może działać w jednym z dwóch trybów: wejścia - oczekiwając na przyjęcie danych od urządzenia zewnętrznego lub wyjścia - wysyłać dane do urządzenia zewnętrznego. Z kolei porty analogowe mogą przesyłać sygnały o większej liczbie stanów logicznych lecz do ich obsługi konieczny jest przetwornik analogowo-cyfrowy, który w Intel Galileo jest 10 bitowy - co oznacza możliwość wysłania jednorazowo do 10 bitów danych.

### **3.0.2. Odpytywanie w pętli**

Jednym z najprostszych metod pozyskania danych z urządzeń wejścia/wyjścia mikro kontrolera jest odpytywanie urządzeń zewnętrznych w nieskończonej pętli. Jest to najmniej efektywny sposób ponieważ zajmuje zasoby sprzętu zapytaniami, które nie zawsze są konieczne. Odczyt stanu urządzeń wejścia/wyjścia może być realizowane mechanizmami systemu operacyjnego (wykorzystując gotowe biblioteki np. Arduino/Wire) lub niskopoziomowo z wykorzystaniem funkcji języka C i/lub Asemblera.

### **3.0.3. Przerwania**

Alternatywnym mechanizmem do odpytywania w pętli jest mechanizm przerwań.

Przerwanie na poziomie procesora jest to sygnał elektryczny pochodzący bezpośrednio z urządzeń do mikroprocesora.

Są to bezpośrednie sygnały systemu lub sprzętu ułatwiające komunikację ze światem zewnętrznym.

System Galileo bazując na procesorze Quark ma standardowy dla procesorów IAPX 86 mechanizm obsługi przerwań obsługiwany za pomocą kontrolera przerwań, którego działanie i konstrukcją jest zbliżona do rozwiązań stosowanych w IBM PC/ATX więc do dyspozycji mamy przerwania:

1. Programowe
2. Sprzętowe
  - (a) Niemaskowalne (NMI<sup>1</sup>)
  - (b) Maskowalne
3. Wyjątek

W momencie gdy zostaje zgłoszone przerwanie wątek programu zostaje zatrzymany po czym wykonywany jest skok do funkcji obsługującej zgłoszone przerwanie.

Przerwania programowe wywołuje się za pomocą instrukcji asemblera *INT XX*, gdzie *XX* oznacza numer przerwania zadeklarowanego w tablicy wektorów przerwań<sup>2</sup>, która jest tworzona przy każdorazowym starcie systemu. W procesorach serii IAPX 86 znajduje się 255 wektorów przerwań.

Przerwanie sprzętowe jest to rodzaj przerwań wywoływanych przez urządzenia wejścia/wyjścia lub zgłaszane przez procesor. Efektem zgłoszenia przerwania sprzętowego jest obsłużenie go poprzez wywołanie przerwania programowego. Przerwania te dzielimy na maskowalne oraz niemaskowalne. Główna

---

<sup>1</sup>Non-Maskable Interrupt

<sup>2</sup>ang. interrupt vector table - tablica zawierająca adresy podprogramów służących do obsługi wektorów przerwań

różnica między nimi polega na możliwości zablokowania przerwań maskowalnych podczas gdy przerwania niemaskowalne muszą zostać obsłużone. Przykładem przerwania niemaskowalnego w systemach IAPX 86 jest *INT2*, który w środowisku Windows znany jest jako popularny *Blue Screen of Death*<sup>3</sup>.

Ostatnim rodzajem przerwań są wyjątki. Wywoływanie są podczas napotkania przez procesor błędów oraz niepowodzeń.

---

<sup>3</sup>Ekran błędu w systemach Windows pojawiający się po krytycznym błędzie systemu

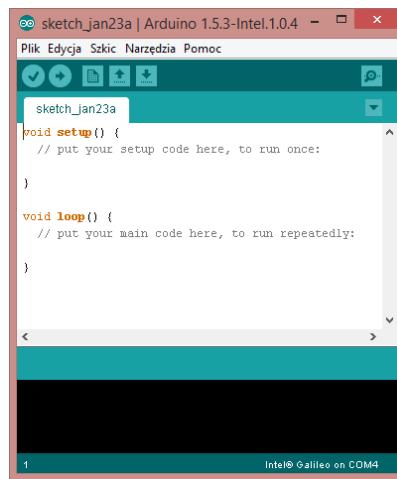
## ROZDZIAŁ 4

# Programowanie Intel Galileo z użyciem różnych środowisk

## 4.1. Programowanie w środowisku Intel Arduino studio

Standardowym środowiskiem programistycznym dla Intel Galileo jest Intel Arduino studio. Przed pierwszym użyciem należy je poprawnie skonfigurować - z zakładki *narzędzia* należy wybrać odpowiednią wersję Intel Galileo oraz podać port pod którym jest ono dostępne w systemie operacyjnym.

Programy pisane w środowisku Arduino różnią się nieznacznie od klasycznych programów pisanych w języku C.



Rysunek 4.1. Arduino Studio

Źródło: Opracowanie własne

Pierwszą różnicą jest jednorazowe wywołanie funkcji *setup()* zaraz po przesłaniu i uruchomieniu programu, która służy do inicjalizacji wszystkich niezbędnych portów. W tej funkcji należy określić początkowe tryby pracy portów. Następnie wywołana zostaje funkcja *loop()*, która jest równoznaczna z funkcją *main()* w języku C, z tą różnicą, że jest wykonywana od momentu startu (zaraz po jednorazowym wykonaniu funkcji *setup()*) aż do wyłączenia systemu.

**Listing 4.1.** Odpowiednik wywołania pętli *loop()* w Arduino dla języka C/C++

```
void main() {
    while(1)
        loop();
    return 0;
}
```

**Listing 4.2.** Odpytywanie funkcji w nieskończonej pętli w środowisku Arduino

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

W środowisku *Arduino* aby obsłużyć port cyfrowy wystarczy ustalić tryb w jakim ma on działać (wejście/wyjście), a następnie wysłać/odczytać dane.

Wpisanie wartości *LOW* jest równoznaczna z wysyłaniem stanu logicznego 0 (napięcie 0V) na określonym pinie, a wpisanie wartości *HIGH* jest równoznaczna z wysyłaniem stanu logicznego 1 (napięcie 5V) na określonym pinie.

Arduino dostarcza szereg funkcji do obsługi portów. Podstawowe z nich to:

1. *pinMode(PIN, MODE);* - funkcja ustawiająca pin o podanym numerze (PIN) na podany tryb pracy - wejście/wyjście (MODE)

20     *Rozdział 4. Programowanie Intel Galileo z użyciem różnych środowisk*

2. *digitalWrite(PIN, VAL);* - funkcja wpisująca wartość (VAL) - HIGH/LOW - do podanego portu cyfrowego (PIN)
3. *digitalRead(PIN);* - funkcja czytająca wartość z podanego portu cyfrowego (PIN)
4. *analogWrite(PIN, VAL);* - funkcja wpisująca wartość (VAL) do podanego portu analogowego (PIN)
5. *analogRead(PIN);* - funkcja czytająca wartość z podanego portu analogowego (PIN)

**Listing 4.3.** Obsługa portu cyfrowego w środowisku Arduino

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

**Listing 4.4.** Obsługa portu analogowego w środowisku Arduino

```
int val = 2;  
int analogPin = A1;  
pinMode(analogPin, INPUT);  
val = analogRead(analogPin);  
pinMode(analogPin, OUTPUT);  
analogWrite(ledPin, val);
```

Arduino obsługuje przerwania. W środowisku Arduino aby zainicjalizować przerwania wystarczy wywołać funkcję *attachInterrupt*:

**Listing 4.5.** Inicjowanie przerwań sprzętowych w środowisku Arduino

```
void setup(){
    attachInterrupt( pinInt , funcName , mode );
}
```

gdzie *pinInt* jest to pin na którym Arduino będzie nasłuchiwało na przerwanie, *funcName* jest to nazwa funkcji, która zostanie wykonana gdy przerwanie zostanie zgłoszone, *mode* jest to określenie kiedy sygnał może być uznany za przerwanie. Należy pamiętać, że funkcja wywoływana przez przerwanie nie może przyjmować żadnych parametrów oraz zwracać żadnego wyniku. *Mode* może przyjmować wartości:

1. LOW - przerwanie zostanie zgłoszone gdy wartość na określonym pinie jest równa LOW
2. CHANGE - przerwanie zostanie zgłoszone gdy wartość na określonym pinie zostanie zmieniona
3. RISING - przerwanie zostanie zgłoszone gdy wartość na określonym pinie zostanie zmieniona z LOW na HIGH
4. FALLING - przerwanie zostanie zgłoszone gdy wartość na określonym pinie zostanie zmieniona z HIGH na LOW

Mechanizmem, który bazuje na przerwaniach jest mechanizm Timera. Polega on na wywołaniu funkcji co określony czas (zgłaszcane jako przerwanie), którego zarządzaniem zajmuje się urządzenie (lub system operacyjny). Rozwiązanie to jest bardzo podobne do odpytywania w pętli, a następnie wywołania funkcji *sleep()* z tą różnicą, że użycie timera jest dokładniejsze ponieważ wykorzystuje zegar czasu rzeczywistego znajdującego się w CPU<sup>1</sup>.

**Listing 4.6.** Przykładowe użycie timera w środowisku Arduino

```
#include <TimerOne.h>
void setup(){
```

---

<sup>1</sup>ang. Central Processing Unit - jednostka arytmetyczno-logiczna wykorzystywana do wykonywania obliczeń niezbędnych do działania programu

```

    Timer1.initialize(500000);
    Timer1.attachInterrupt(funcName, 500000);
}

```

## 4.2. Komunikacja z urządzeniami poprzez mechanizmy systemu operacyjnego Linux YOCTO

Z punktu widzenia systemu operacyjnego Linux YOCTO urządzenia wejścia/wyjścia są traktowane tak jak pliki. W systemie YOCTO Linux dostępne są pliki w katalogu `/sys/class/gpio/` odpowiadające poszczególnym portom w Galileo. Przy komunikacji należy jednak pamiętać, że nazwy urządzeń nie są intuicyjne np. port IO4 nie jest plikiem `/sys/class/gpio/gpio4` (Patrz Dodatek C). Komunikację z portami można obrazować jako wpisanie lub odczytanie danych z pliku. Na początku należy ustalić w jakim trybie ma działać port. W tym celu do pliku `/sys/class/gpio/PORT/direction` należy wpisać wartość `out` dla ustawienia jako wyjście lub `in` dla ustawienia jako wejście. Następnie można odczytać lub wpisać dane do wcześniej skonfigurowanego portu. W tym celu do pliku `/sys/class/gpio/PORT/value` należy wpisać wartość `1` dla ustawienia stanu wysokiego lub `0` dla ustawienia stanu niskiego, gdzie `PORT` jest to numer odpowiedniego portu według numeracji Galileo. Warto zauważyć, że przesyłane wartości tym mechanizmem nie są wartościami 8 lecz 1 bitowymi.

**Listing 4.7.** Obsługa portu cyfrowego w środowisku Linux za pomocą języka C/C++

```

FILE *fp ;
int value ;

// Ustawienie portu cyfrowego nr 13 jako port wyjścia
fp = fopen( "/sys/class/gpio/gpio39/direction" , "w" );

```

```
fprintf(fp, "out");
fclose(fp);

// Wpisanie wartosci do portu cyfrowego
fp = fopen("/sys/class/gpio/gpio39/value", "w");
fprintf(fp, "1");
fclose(fp);

// Odczytanie wartosci z portu cyfrowego
fp = fopen("/sys/class/gpio/gpio39/value", "r");
fscanf(fp, "%i", &value);
fclose(fp);
```

oraz podobnie dla języków skryptowych np. Bash:

**Listing 4.8.** Obsługa portu cyfrowego w środowisku Linux (bash)

```
# Ustawienie portu cyfrowego nr 13 jako port swijcia
root@henio:~# echo -n "out" > /sys/class/gpio/gpio39/direction

# Wpisanie wartosci do portu cyfrowego
root@henio:~# echo -n "0" > /sys/class/gpio/gpio39/value
root@henio:~# echo -n "1" > /sys/class/gpio/gpio39/value

# Odczytanie wartosci z portu cyfrowego
root@henio:~# echo -n "in" > /sys/class/gpio/gpio28/direction
root@henio:~# cat /sys/class/gpio/gpio28/value
```

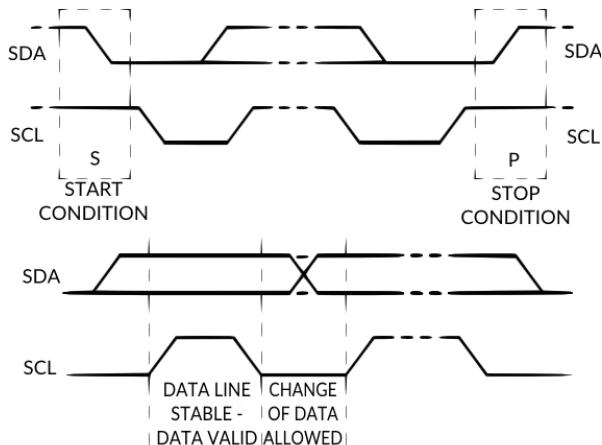
Należy pamiętać aby najpierw ustalić tryb w jakim ma działać port. W przypadku ustawienia portu w tryb wyjścia - *OUT* - gdy nie zostanie to zrobione przed próbą wpisania wartości to otrzymamy błąd: *write error: Operation not permitted*.

## ROZDZIAŁ 5

# Implementacja

### 5.1. Protokół komunikacyjny $I^2C$

Protokół komunikacyjny  $I^2C$ <sup>1</sup> jest szeregowym interfejsem stworzonym przez firmę *Philips* służącym do przesyłania danych między urządzeniami elektrycznymi. W projekcie protokół ten został użyty do komunikacji z I/O Expander PCF 8574N obsługujący płytke symulatora samochodu.



Rysunek 5.1. Przebieg czasowy protokołu  $I^2C$

Źródło: <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/> [6]

Podstawową cechą  $I^2C$  jest wykorzystywanie jedynie dwóch linii służących do komunikacji: dwukierunkowa linia danych  $SDA$ <sup>2</sup> oraz jednokierun-

<sup>1</sup>ang. Inter-Integrated Circuit

<sup>2</sup>ang. Serial Data Line

kowa linia zegarowa  $SCL$ <sup>3</sup>. Protokół  $I^2C$  bazuje na przesyłaniu ramek (pakiętów) składających się z sekwencji: START -> adres -> dane -> STOP.

W celu wygenerowania impulsu START należy ustawić linię  $SDA$  oraz  $SCL$  w stan  $HIGH$  (5V) po czym w trakcie gdy linia  $SCL$  jest w stanie  $HIGH$  należy zmienić stan linii  $SDA$  na stan  $LOW$ . Analogicznie do wygenerowania impulsu STOP należy w trakcie gdy linia  $SCL$  jest w stanie  $HIGH$  zmienić stan linii  $SDA$  ze stanu  $LOW$  na stan  $HIGH$ . Dane wysyłane/odbierane są bit po bicie - na początku należy ustawić linię  $SCL$  w stan wysoki (HIGH), odczytać wartość na linii  $SDA$ , a następnie zmienić stan linii  $SCL$  na niski (LOW).

Adresowanie urządzenia odbywa się poprzez wysłanie pojedynczych bitów adresu (pamiętając o kolejności MSB->LCB<sup>4</sup>) oraz wygenerowanie impulsu zegara. Gdy chcemy zaadresować urządzenie, którego adresem jest np. 4 należy wykonać:

**Listing 5.1.** Własna wersja adresowania urządzenia  $I^2C$  na przykładzie PCF8574N

```
int adres = 4;
for (m = 0x80; m; m >>= 1){
    if (adres & m)
        digitalWrite(sda, HIGH);
    else
        digitalWrite(sda, LOW);

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

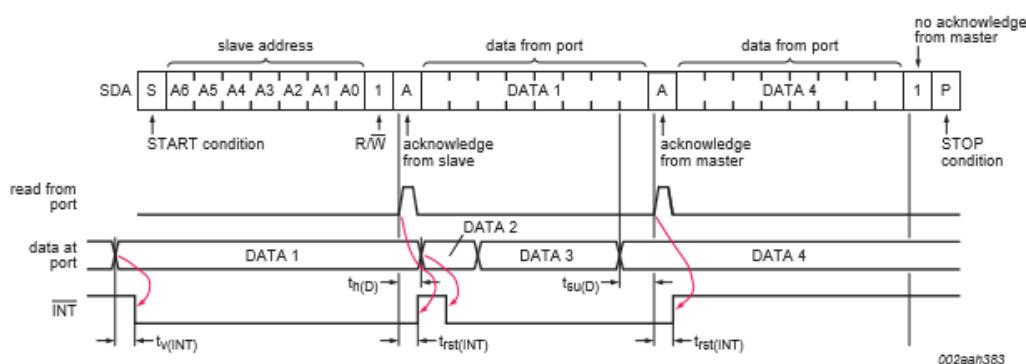
Należy pamiętać, że część adresu jest już podana przez producenta w dokumentacji i jest specyficzna dla danego modelu sprzętu. Jednak nie zawsze dokumentacja jest dostatecznie jasna dla użytkownika i można się spotkać

<sup>3</sup>ang. Serial Clock Time

<sup>4</sup>Wysyłanie odbywa się w kolejności od najbardziej znaczących (najstarszych) bitów

z nieścisłościami w postaci różnic podanych np. w tabelach i wykresach czasowych. Wtedy pozostaje tylko metoda prób i błędów.

Kolejnym krokiem jest ustalenie czy będziemy chcieli z urządzeniem przeczytać dane czy je wysłać. W tym celu należy wysłać 1 lub 0 jako kolejny bit. Po otrzymaniu potwierdzenia na linii *SDA* można zacząć komunikację z urządzeniem. Na zakończenia transmisji należy wysłać sygnał *STOP*.



**Rysunek 5.2.** Schemat odbierania/wysyłania danych poprzez  $I^2C$  na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

Dane wysyłane są od najstarszego do najmłodszego bitu. Każda paczka potwierdzona jest przez odbiornik (bit ACK<sup>5</sup>). Należy również pamiętać, aby każdą komunikację z urządzeniem rozpocząć i zakończyć ustawiając linie *SDA* oraz *SCL* w stan nieaktywny (HIGH) - zgodnie z prawidłowym wygenerowaniem impulsu STOP.

**Listing 5.2.** Odebranie danych z użyciem protokołu  $I^2C$  na przykładzie PCF8574N

```
int readPCF(char adres){
```

```
    int m, ack, wynik = 0;
```

```
    // Deklaracja potrzebnych zmiennej
```

<sup>5</sup>ang. Acknowledge

```
digitalWrite(sda, LOW);
delay(d);
digitalWrite(scl, LOW);
for(m = 0x80; m; m >>= 1){      // Adresowanie urządzenia
    if(adres & m)
        digitalWrite(sda, HIGH);
    else
        digitalWrite(sda, LOW);
    digitalWrite(scl, HIGH);          // Wygenerowanie impulsu zegara
    delay(d);
    digitalWrite(scl, LOW);
    delay(d);
}
pinMode(sda, INPUT);
digitalWrite(scl, HIGH);
delay(d);
ack = digitalRead(sda);           // Odczytanie bitu ACK
digitalWrite(scl, LOW);
delay(d);
for(m = 0x80; m; m>>=1){      // Odczytanie danych z urządzenia
    digitalWrite(scl, HIGH);
    delay(d);
    if(digitalRead(sda)){
        wynik |= m;
    }
    digitalWrite(scl, LOW);
    delay(d);
}
pinMode(sda, OUTPUT);             // Wygenerowanie bitu STOP
digitalWrite(scl, HIGH);
delay(d);
digitalWrite(sda, HIGH);
```

```

delay(d);
return wynik;           // Zwrócenie wyniku
}

```

Podstawowymi zaletami protokołu są:

1. Połączenia składają się tylko z dwóch linii co znacznie ogranicza liczbę kabli wychodzących z urządzenia
2. Częstotliwość pracy wynosi 400kHz
3. Dane przesyłane są w kolejności MSB->LSB
4. Każde urządzenie ma swój adres
5. Transmisja jest odporna na zakłócenia zewnętrzne

Nazwa  $I^2C$  jest nazwą zastrzeżoną przez firmę *Philips* dlatego też w literaturze bardzo często spotyka się określenie *TWI*<sup>6</sup>. Jest ono stosowane w mikro kontrolerach firmy *Atmel*.

W środowisku Arduino dostępna jest biblioteka do obsługi  $I^2C$  Wire jednak podczas próby użycia jej w projekcie wystąpiły problemy z kompatybilnością z używanym sprzętem w związku z tym na potrzeby projektu napisana została własna wersja biblioteki obsługującej komunikację poprzez protokół  $I^2C$ .

### 5.1.1. I/O Expander PCF8574N

W symulatorze samochodu ze względu na dużą ilość wychodzących sygnałów zastosowano I/O Expander wykorzystujący komunikację poprzez  $I^2C$ . Do obsługi tych sygnałów zostały zamontowane dwa I/O Expander PCF 8574N zbierające wszystkie sygnały cyfrowe wychodzące z symulatora do Galileo (z powodów technicznych I/O Expander został później zastąpiony na PCF 8574AN). Dzięki temu zamiast używać dwunastu linii cyfrowych, wykorzystywane są jedynie dwie niezbędne do komunikacji poprzez  $I^2C$  (*SDA, SCL*)

---

<sup>6</sup>ang. Two Wire Interface

co znacznie ułatwiło korzystanie z Galileo ze względu na pozostałe wolne porty cyfrowe, które zostały wykorzystane w dalszej części pracy do komunikacji z wyświetlaczem i innymi elementami simulatora. Zaletą tego rozwiązania jest możliwość późniejszego podłączenia większej ilości czujników bez jakiegokolwiek ingerencji w okablowanie Galileo.

## 5.2. Protokół komunikacyjny SPI

Protokół SPI<sup>7</sup> jest szeregowym protokołem komunikacyjnym składającym się z czterech podstawowych linii - dwóch służących do przesyłania danych w przeciwnych kierunkach, jednej z sygnałem taktującym, synchronizującym transfer danych oraz linii *Chip Select*. *Chip Select* jest odpowiednikiem adresu urządzenia z protokołu  $I^2C$ . Każde urządzenie podłączone pod system musi mieć osobną linię *Chip Select*. Gdy linia zostanie aktywowana (ustawiona w stan LOW) można rozpoczęć komunikację z wybranym urządzeniem.

Linia MISO<sup>8</sup> jest linią wejścia danych dla urządzenia nadzawanego (master), a wyjściem dla urządzenia podrzędnego (slave), linia MOSI<sup>9</sup> jest wyjściem dla urządzenia master, a wejściem dla slave. Linia SCK<sup>10</sup> jest wejściem taktującym zegar. Sygnał taktujący jest zawsze generowany przez układ master. 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.<sup>[7]</sup>

Gdy chcemy wysłać dane 8 bitowe poprzez protokół SPI należy: aktywować odpowiednią linię *Chip Select*, a następnie wysłać dane. Gdy mamy do dyspozycji porty cyfrowe należy to zrobić bit po bicie.

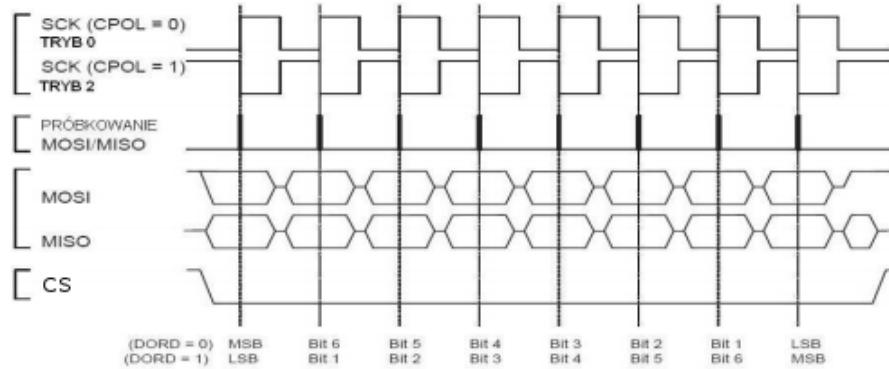
**Listing 5.3.** Wysłanie danych 8 bitowych za pomocą protokołu SPI

<sup>7</sup>ang. Serial Peripheral Interface

<sup>8</sup>ang. Master In Slave Out

<sup>9</sup>ang. Master Out Slave In

<sup>10</sup>ang. Serial Clock



**Rysunek 5.3.** Przebiegi czasowe interfejsu SPI

Źródło: <http://castor.am.gdynia.pl/~dorra>[7]

```

void sendData(int data, int CS, int clock, int SDO){
    digitalWrite(CS, LOW);
    int i;
    for( i = 0x80; i; i >>= 1){
        digitalWrite(SDO, data & i);           // Wyisanie bitu
        digitalWrite(clock, HIGH);   // Wygenerowanie impulsu zegara
        digitalWrite(clock, LOW);
    }
    digitalWrite(CS, HIGH);
}

```

### 5.2.1. Komunikacja z ekranem FTDI EVE VM800b poprzez protokół SPI

Wyświetlacz TFT FTDI EVE VM800B jest obsługiwany za pomocą protokołu SPI. Podobnie jak w przypadku protokołu  $I^2C$  została napisana własna wersja biblioteki do obsługi protokołu SPI. Do komunikacji niezbędne było napisanie funkcji implementujących niskopoziomowe procedury komunikacyjne:

1. *void sendData(int data)* - funkcja wysyłająca wartość 8 bitową
2. *void ft800memWriteX(unsigned long ftAddress, unsigned char ftDataX)*  
- funkcja wpisująca podaną wartość X - bitową na podany adres, gdzie X może być wartością 8, 16 lub 32 bitową
3. *unsigned char ft800memReadX(unsigned long ftAddress)* - funkcja odczytująca wartość X - bitową z podanego adresu, gdzie X może być wartością 8, 16 lub 32 bitową
4. *unsigned int incCMDOffset(unsigned int currentOffset, unsigned char commandSize)* - funkcja zwiększająca offset w buforze pamięci ekranu
5. *void ft800cmdWrite(unsigned char ftCommand)* - funkcja wysyłająca podaną komendę do ekranu - np. Start urządzenia (*FT800ACTIVE*)

Wyświetlacz ma możliwość korzystania z systemu *HMI* więc do obsługi zostało napisane proste API, które dostarcza następujące funkcjonalności:

1. inicjalizacja ekranu
2. rysowanie kółka o podanym rozmiarze, kolorze i w podanym miejscu
3. rysowanie linii o podanych końcach, szerokości oraz kolorze
4. wypisanie tekstu
5. wypisanie cyfr
6. narysowanie guzika w podanym miejscu i podanym rozmiarze

Przykładowe wyświetlenie linii o pozycji, kolorze oraz szerokości podanej w parametrach:

**Listing 5.4.** Narysowanie linii na ekranie

```
void linia(unsigned long color, unsigned long x1, unsigned long y1,  
          unsigned long x2, unsigned long y2, unsigned long width){  
    ft800memWrite32(RAM_CMD+cmdOffset, (DL_BEGIN|LINES));
```

```

cmdOffset=incCMDOffset(cmdOffset , 4);

ft800memWrite32(RAM_CMD+cmdOffset , (DL_COLOR_RGB|color));
cmdOffset=incCMDOffset(cmdOffset ,4);

ft800memWrite32(RAM_CMD+cmdOffset , (DL_LINE_WIDTH|width));
cmdOffset=incCMDOffset(cmdOffset ,4);

ft800memWrite32(RAM_CMD+cmdOffset , (DL_VERTEX2F|(x1<<15)|y1));
cmdOffset=incCMDOffset(cmdOffset ,4);

ft800memWrite32(RAM_CMD+cmdOffset , (DL_VERTEX2F|(x2<<15)|y2));
cmdOffset=incCMDOffset(cmdOffset ,4);
}

```

Podobnie gdy chcemy wyświetlić na ekranie kropkę:

**Listing 5.5.** Narysowanie kropki na ekranie

```

void kropka(unsigned long color , unsigned int size , unsigned long
unsigned long y){
    ft800memWrite32(RAM_CMD+cmdOffset , (DL_POINT_SIZE|size));
    cmdOffset=incCMDOffset(cmdOffset ,4);

    ft800memWrite32(RAM_CMD+cmdOffset , (DL_BEGIN|FTPOINTS));
    cmdOffset=incCMDOffset(cmdOffset , 4);

    ft800memWrite32(RAM_CMD+cmdOffset , (DL_COLOR_RGB|color));
    cmdOffset=incCMDOffset(cmdOffset ,4);

    ft800memWrite32(RAM_CMD+cmdOffset , (DL_VERTEX2F|(x<<15)|y));
    cmdOffset=incCMDOffset(cmdOffset ,4);
}

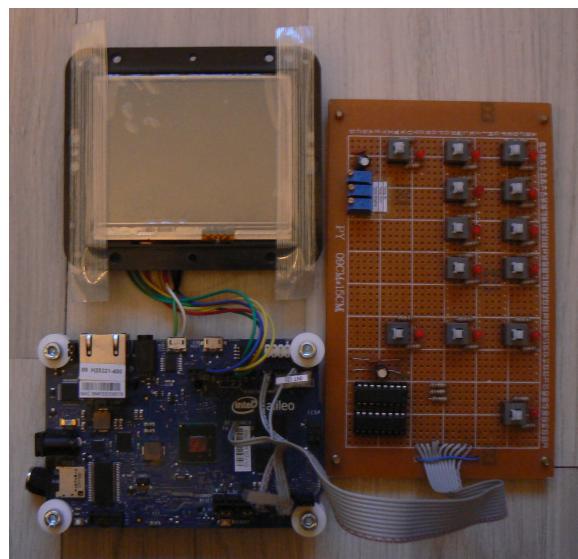
```

## **ROZDZIAŁ 6**

# **Działanie komputera pokładowego**

## **6.1. Założenia funkcjonalne projektu**

Najważniejszym założeniem funkcjonalnym projektowanego systemu była komunikacja z zestawem czujników, które mogą być zamontowane w samochodzie. W projekcie zostały użyte czujniki otwarcia/zamknięcia drzwi, zapięcia pasów, włączenia/wyłączenia światel oraz czujniki temperatury. Dodatkowym elementem była komunikacja z zewnętrznym wyświetlaczem TFT służącym do komunikacji pomiędzy użytkownikiem a komputerem. Samochód zastąpiony został za pomocą zbudowanego symulatora, którego schemat elektryczny został przedstawiony na rysunku 6.3.

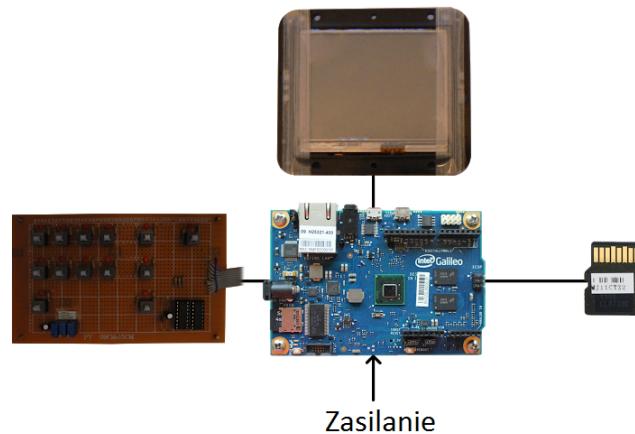


**Rysunek 6.1.** Zdjęcie gotowego zestawu

Źródło: Opracowanie własne

Port Arduino	Opis	Typ	Połączenie
2	Przerwania	Cyfrowy	Symulator samochodu
6	Linia SCL	Cyfrowy	Symulator samochodu
7	Linia SDA	Cyfrowy	Symulator samochodu
8	Linia SDI	Cyfrowy	Ekran FTDI EVE VM800B
9	Linia SDO	Cyfrowy	Ekran FTDI EVE VM800B
10	Linia clock	Cyfrowy	Ekran FTDI EVE VM800B
11	Linia PD	Cyfrowy	Ekran FTDI EVE VM800B
12	Linia CS	Cyfrowy	Ekran FTDI EVE VM800B
14	GROUND	Cyfrowy	Ekran FTDI EVE VM800B
A0	Temperatura w środku	Analogowy	Symulator samochodu
A1	Temperatura na zewnątrz	Analogowy	Symulator samochodu
A2	Temperatura silnika	Analogowy	Symulator samochodu

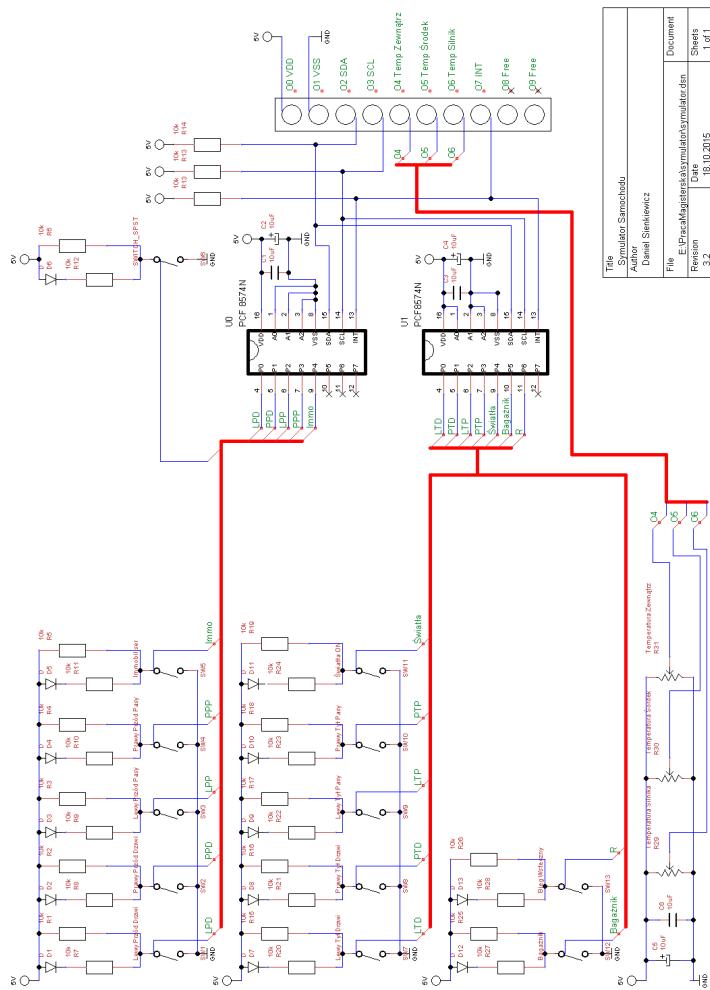
**Tabela 6.1.** Porty użyte do działania systemu



**Rysunek 6.2.** Schemat logiczny zestawu

Źródło: Opracowanie własne

Symulator samochodu został wykonany za pomocą 12 przełączników binstabilnych. Stan każdego przełącznika sygnalizowany jest za pomocą diody LED. Sygnały ze wszystkich przełączników odczytywane są za pomocą dwóch I/O Expanderów PCF8547. Do symulacji czujników temperatury zamontowane zostały trzy potencjometry wysyłające sygnał analogowy. Wszystkie sygnały doprowadzone zostały do wtyczki, która poprzez taśmę podłączona jest do Intel Galileo. Cały zestaw zasilany jest przez kontroler napięciem 5V.



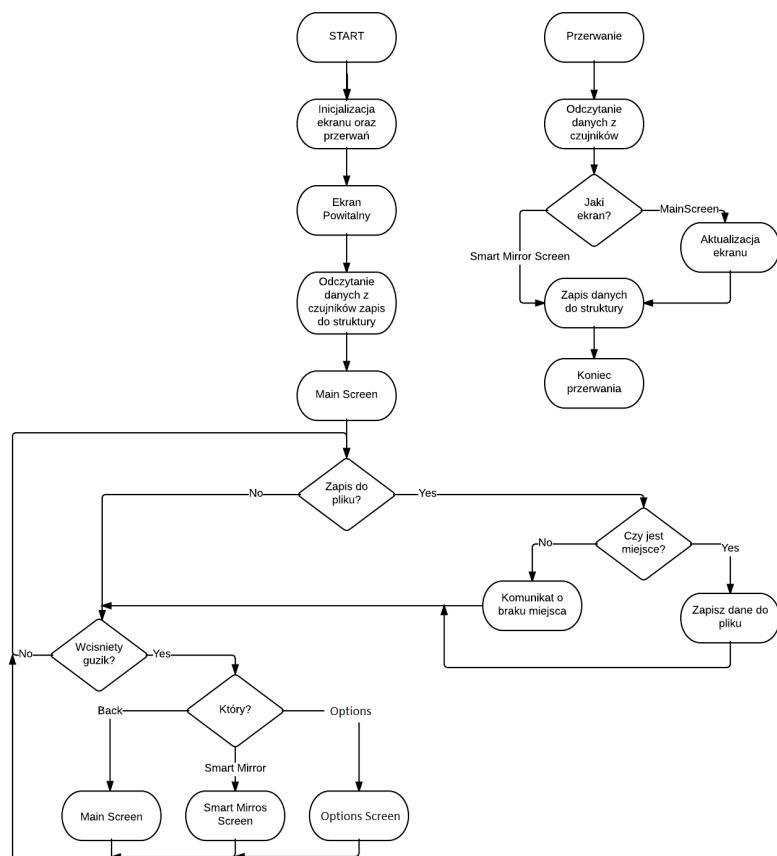
Rysunek 6.3. Schemat elektryczny symulatora samochodu

Źródło: Opracowanie własne

## 6.2. Opis budowy i działania

Proponowany komputer pokładowy jest całkowicie osobnym systemem niezależnym od sprzętu aktualnie posiadanego w samochodzie.

Schemat działania systemu został przedstawiony na schemacie 6.4. Obsługa przycisków została zrobiona za pomocą odpytywania w nieskończonej pętli, odczyt z czujników temperatury jest wykonywany na podstawie timera systemowego a aktualizacja drzwi, pasów oraz świateł jest wykonywana po otrzymaniu przerwania sprzętowego z symulatora samochodu.



Rysunek 6.4. Schemat blokowy działania programu

Źródło: Opracowanie własne

Po włączeniu systemu na ekranie pojawia się powitanie oraz ekran startowy, na którym można zobaczyć symulację aktualnej pozycji *GPS*, temperaturę panującą w środku samochodu, na zewnątrz oraz w silniku.



**Rysunek 6.5.** Ekran startowy

Źródło: Opracowanie własne

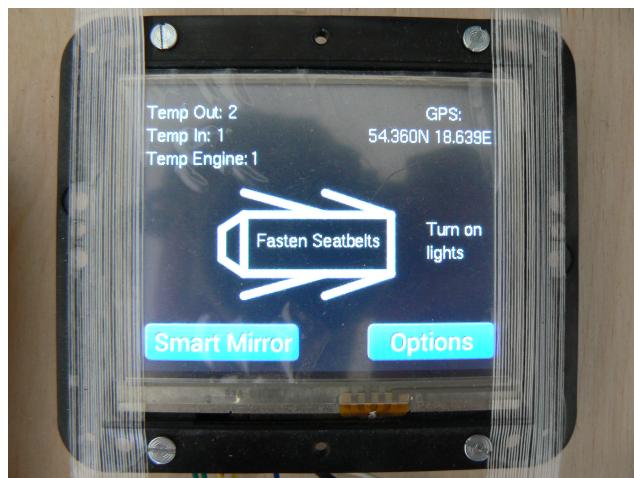
Stan drzwi i pasów z wizualizowany został poprzez miniaturkę samochodu z aktywnie otwierającymi się drzwiami, zapalającymi światłami oraz ikonką obrazującą stan zapiętych pasów.

Na ekranie głównym znajdują się 2 przyciski:

1. Smart Mirror
2. Options

Pierwszy z nich umożliwia przejście w tryb aktywnego lusterka wstecznego, który może również służyć jako czujnik cofania co może przydać się podczas parkowania w ciasnych miejskich. Drugi pozwala na przejście do ekranu opcji systemu.

W opcjach znajduje się możliwość rozpoczęcia zapisu danych dotyczących stanu samochodu na karcie pamięci wraz z wybranym jednym z trzech



Rysunek 6.6. Ekran główny

Źródło: Opracowanie własne

formatów zapisu danych - JSON<sup>1</sup>, XML<sup>2</sup> oraz CSV<sup>3</sup>. W tym celu zostały zaimplementowane funkcje realizujące konwersję danych do odpowiedniego formatu. Po wybraniu opcji zapisu danych, będą one zapisywane do momentu jej wyłączenia lub wyczerpania miejsca na karcie pamięci. Interwał zapisu jest również jedną z dostępnych opcji wyboru - standardowo użytkownik ma do wyboru zapis co 5s, 10s, 30s, 1 min, 15 min.

<sup>1</sup>ang. JavaScript Object Notation - jest prostym formatem wymiany danych. Jego definicja opiera się o podzbiór języka programowania JavaScript, Standard ECMA-262 3rd Edition - December 1999. JSON jest formatem tekstowym, całkowicie niezależnym od języków programowania, ale używa konwencji, które są znane programistom korzystającym z języków z rodziny C, w tym C++, Java, JavaScript, Perl, Python i wielu innych. [8]

<sup>2</sup>ang. Extensible Markup Language. Wywodzi się od języka SGML i jest językiem znaczników służącym do opisu danych. Dane przechowywane są w postaci tekstu w dokumencie o ścisłe określonej strukturze. XML jest standardem przemysłowym i stosowany jest we wszystkich dziedzinach informatyki.[9]

<sup>3</sup>ang. Comma-separated values. Format przechowywania danych w postaci tekstu. W pierwszej linii pliku znajdują się oddzielone przecinkami nazwy danych jakie są przechowywane. W kolejnych liniach wpisane są odpowiednie wartości w kolejności ustalonej przez pierwszą linię.

Wyłączenie systemu następuje wraz z wyłączeniem zapłonu w samochodzie.



**Rysunek 6.7.** Ekran opcji

Źródło: Opracowanie własne

Komunikacja z ekranem TFT odbywa się za pomocą protokołu *SPI*. Łącznie użytych zostało 7 linii: *SDI*, *SDO*, *clock*, *PD*, *CS*<sup>4</sup> oraz *PWR*<sup>5</sup> i *GND*<sup>6</sup>, natomiast komunikacja komputera z symulatorem odbywa się za pomocą protokołu *I<sup>2</sup>C*. Odczytanie wartości czujników cyfrowych odbywa się po otrzymaniu przerwania sprzętowego zgłaszanego przez I/O Expander przy zmianie jakiejkolwiek wartości np. przy otworzeniu drzwi. Odczyt czujników analogowych (temperatura) odbywa się przy użyciu timera co określony w programie czas.

Gdy została wybrana opcja zapisu danych na kartę pamięci wtedy w nieskończonej pętli w odstępach czasu wybranych przez użytkownika aktualna pozycja *GPS* zostaje zapisywana do pliku tekstowego do momentu wyłączenia systemu, wyczerpania miejsca na karcie lub zmiany tej opcji przez użytkownika. Zapis do pliku odbywa się poprzez użycie standardowych funkcji

<sup>4</sup>Linia Chip Select wyświetlacza

<sup>5</sup>ang. Power - 5V

<sup>6</sup>ang. Ground

języka C znajdujących się w *stdlib.h*. Możliwy jest w 3 najbardziej popularnych formatach: CSV, JSON oraz XML. Format może zostać wybrany przez użytkownika poprzez zmianę w ustawieniach komputera.

**Listing 6.1.** Obsługa karty microSD za pomocą mechanizmu Arduino

```
void writeSD( char *filename , char * data ){
    File dataFile = SD.open( filename , FILE_WRITE );
    dataFile . println( data );
    dataFile . close ();
}

void readSD( char *filename ){
    File dataFile = SD.open( filename );
    while ( dataFile . available () ) {
        Serial . write ( dataFile . read () );
    }
    dataFile . close ();
}
```

**Listing 6.2.** Obsługa karty microSD za pomocą mechanizmu systemu operacyjnego

```
void writeSD (){
    String command = "";
    command = "echo \u0144tekst";
    command += "\u0144>\u0144/tmp/daniel.txt";
    system ( command . buffer );
}
```

**Listing 6.3.** Obsługa karty microSD za pomocą języka C

```
FILE *fp ;
fp = fopen ( "/tmp/daniel.txt" , "a" );
fprintf ( fp , "tekst" );
fclose ( fp );
```

Intel Galileo został wyposażony w zegar czasu rzeczywistego (RTC), który został wykorzystany do synchronizacji zapisu danych do pliku. Dane zapisywane są do pliku, którego nazwa jest taka sama jak data. Do odczytania aktualnej daty wystarczy użyć standardowych funkcji dostępnych w języku C. Należy jednak pamiętać o tym, aby zegar był cały czas zasilany poprzez zewnętrzną baterię. W przeciwnym wypadku za każdym razem będzie on resetowany do domyślnej daty.

**Listing 6.4.** Odczyt aktualnej daty

```
system( "date \u201e+\u00d9H:\u00d9M:\u00d9S \u201c>\u201e/tmp/time.txt" );
char buf[9];
FILE *fp;
fp = fopen( "/tmp/time.txt" , "r" );
fgets( buf , 9 , fp );
fclose( fp );
```

W systemie samochód obrazowany jest za pomocą struktury, która jako pola posiada informację na temat aktualnego stanu pojazdu. Drzwi oraz pasy zostały zobrazowane jako liczba naturalna, w której poszczególne bity oznaczają stany binarne (1 - otwarcie, 0 - zamknięte).

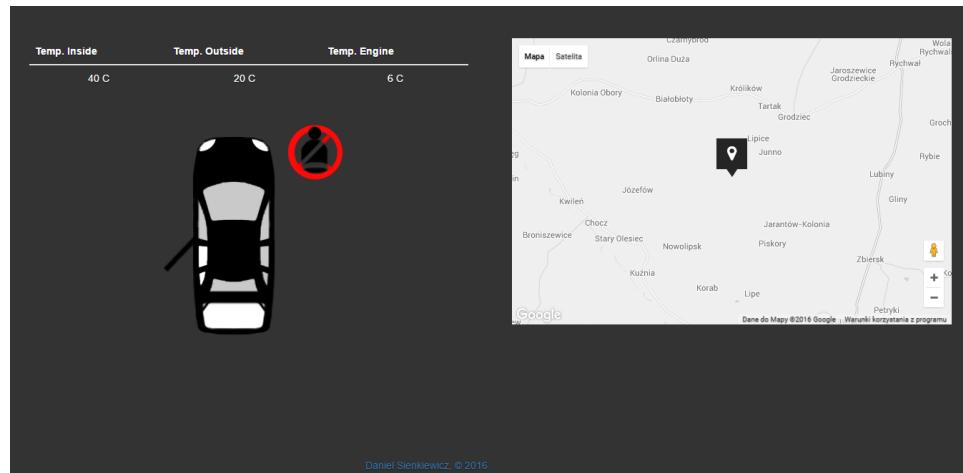
**Listing 6.5.** Struktura samochodu w programie

```
struct car {
    int doors;
    int seatbelts;
    int lights;
    int r;
    float tempOut;
    float tempIn;
    float tempEngine;
};
```

Dodatkowym modułem systemu jest uruchomiony na nim napisany w technologii *NodeJS* serwer www uruchomiony na mikrokomputerze Intel Galileo udostępniający usługę śledzenia pojazdu. Gdy system zostanie podłączony do internetu (np. poprzez wejście Ethernet lub kartę Wi-Fi) użytkownik ma możliwość wejścia na stronę www gdzie może na bieżąco sprawdzać położenie samochodu na mapie oraz aktualną temperaturę wraz ze stanem drzwi oraz pasów. Komunikacja systemu z serwerem odbywa się poprzez udostępnione *REST API*. Użycie technologii *AngularJS* pozwala na wyświetlanie bieżących danych bez konieczności odświeżania strony.

**Listing 6.6.** Request wysyłany do serwera www

```
curl -i -X POST -H 'Content-Type:application/json' -d
'{"tempIn": "40", "tempOut": "20", "tempEngine": "6",
"GPSlongitude": "52", "GPSlatitude": "18",
"doors": "5", "seatbelt": "6", "lights": "0"}'
http://localhost:3000/updateData
```



**Rysunek 6.8.** Strona www

Źródło: Opracowanie własne

## 6.3. Wnioski oraz własne doświadczenia

Komputer został tak zaprojektowany tak aby w łatwy sposób można było dodać kolejne funkcjonalności zależne od potrzeb użytkownika. Jako propozycje można uwzględnić:

1. Lokalizator *GPS* - wczytywanie aktualnej pozycji *GPS* i wyświetlanie jej na wyświetlaczu
2. Kamerka cofania - wyświetlanie obrazu z kamerki cofania na wyświetlaczu
3. Czujnik deszczu - automatyczne włączenie wycieraczek i dopasowanie ich prędkości w zależności od obfitości opadów i prędkości samochodu, dodatkowe włączenie wycieraczki tylnej w momencie gdy zostanie wrzucony bieg wsteczny
4. Sterowanie głośnością radia w zależności od prędkości samochodu
5. Blokada immobilizer
6. Obsługa telefonu komórkowego za pomocą bluetooth
7. Router - dodanie modułu karty WiFi w połączeniu z odbieraniem sieci komórkowej *GPRS* oraz rozsyłanie jej w samochodzie

Jednak na potrzeby tej wersji projektu nie zostały one zaimplementowane. Oczywiście ogranicza nas tylko nasza wyobraźnia oraz finanse jakie chcemy przeznaczyć na rozbudowę systemu o dodatkowe moduły.

Intel Galileo ma wiele zalet ale niestety posiada również i wady. Największą zaletą jest możliwość obsługi systemu operacyjnego Linux oraz standard pinów *GPIO* w pełni kompatybilny z popularnym Arduino co daje możliwość zakupu wielu dodatkowych dedykowanych modułów. Posiadanie pełnego systemu operacyjnego daje możliwość wykorzystania go jako na przykład serwer www lub domowego serwera multimedialnego.

Pierwszą rzeczą na którą należy uważać jest zasilanie. Galileo ma możliwość zasilania poprzez port *USB* lecz w praktyce nie jest to możliwe. Podczas uruchomienia prąd dostarczany poprzez *USB* jest nie wystarczający i następuje automatyczne wyłączenie sprzętu podczas którego Software zostaje uszkodzony. Kolejną wadą jest szybkość komunikacji poprzez piny *GPIO*. Są one w porównaniu do innych urządzeń dostępnych na rynku bardzo wolne co znacznie ogranicza jego możliwości.

Podczas pracy nad projektem niestety uszkodzone zostało Galileo. Po zamianie na Galileo Gen 2 i przetestowaniu tego samego programu okazało się, że w wersji 2 znacznie poprawiona zastała obsługa portów *GPIO*. Program, który na Gen 1 uruchamiał się w 40s, na Gen 2 uruchamia się 4s.

# Zakończenie

Celem niniejszej pracy było wykonanie komputera pokładowego do samochodu bazującego na mikrokomputerze Intel Galileo oraz sprawdzenie możliwości samego mikrokomputera. Przedstawiony projekt nie jest w pełni kompletnym rozwiązaniem. Funkcjonalności takie jak określanie pozycji GPS oraz inteligenckie lusterko wsteczne zostały jednak za symulowane na potrzeby pracy.

Układ komputera został zaprojektowany i zbudowany przy użyciu mikrokomputera Intel Galileo oraz ekranu FTDI EVE VM800B. Układ symulatora samochodu został zaprojektowany i przedstawiony w postaci schematu elektrycznego przy pomocy, którego został on później zbudowany.

Do działania całego systemu zostało napisane własne oprogramowanie wraz z niezbędnymi bibliotekami służącymi do komunikacji z urządzeniami wejścia/wyjścia poprzez użycie protokołów  $I^2C$  oraz  $SPI$ . Użycie ekranu bazuje na napisanym zestawie funkcji umożliwiającym wykorzystanie obsługiwanej *HMI*.

Jedną z możliwości systemu był zapis danych na kartę pamięci *microSD*, który został oprogramowany przy użyciu własnych funkcji z wykorzystaniem bibliotek systemu operacyjnego.

Podczas pracy nad projektem udało się sprawdzić możliwości Intel Galileo wykorzystując różne interfejsy komunikacyjne (GPIO, Ethernet, USB, RS-232) oraz możliwości systemu *Linux YOCTO*. Przetestowana została również możliwość skompilowania i uruchomienia programu z pominięciem standardowego środowiska Intel Arduino Studio i komunikacji poprzez USB. W tym celu wykorzystane zostały możliwości systemu operacyjnego Linux takie jak: protokół *SSH* oraz kompilator *GCC*.

Dodatkową funkcjonalnością było napisane systemu umożliwiającego śledzenie pojazdu oraz aktualnych informacji o nim poprzez sieć Ethernet z wykorzystaniem technologii *NodeJS* oraz *AngularJS*.

Dokumentacja kodu została napisana za pomocą notacji wykorzystanej w systemie *DOXYGEN* i zapisana do pliku *PDF*.

Reasumując Intel Galileo jest bardzo ciekawym rozwiązaniem w świecie IoT<sup>7</sup>. Posiada ogromne możliwości do wykorzystania jako na przykład domowy serwer multimedialny lub kontroler urządzeń wejścia/wyjścia. Osobiście polecałbym generację 2 ze względu na poprawioną obsługę portów GPIO co znacznie ułatwia korzystanie z nich oraz znacznie wpływa na wydajność komunikacji pomiędzy urządzeniami.

---

<sup>7</sup>ang. Internet of things - system powiązanych ze sobą urządzeń gromadzących i przetwarzających informacje najczęściej za pośrednictwem sieci komputerowej

## **DODATEK A**

### **Karty Katalogowe**

Katalog *datasheets* zawiera karty katalogowe użytych podzespołów

1. Intel Galileo.pdf - Karta katalogowa Intel Galileo
2. PCF8574.pdf - Karta katalogowa I/O Expander PCF 8574N
3. VM800B.pdf - Karta katalogowa ekranu FTDI EVE VM800B

## 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	700MHz	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/>[10]

Źródło: <http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html>[11]

## DODATEK C

# Mapowanie portów Intel Galileo na pliki w systemie Linux

Quark X1000	Sysfs GPIO	Galileo/Arduino port
GPORT4 BIT7	gpio51	IO1
GPIO6	gpio14	IO2
GPIO7	gpio15	IO3
GPORT1 BIT4	gpio28	IO4
GPORT0 BIT1	gpio17	IO5
GPORT1 BIT0	gpio24	IO6
GPORT1 BIT3	gpio27	IO7
GPORT1 BIT2	gpio26	IO8
GPORT0 BIT3	gpio19	IO9
GPORT0 BIT0	gpio16	IO10
GPORT1 BIT1	gpio25	IO11
GPORT3 BIT2	gpio38	IO12
GPORT3 BIT3	gpio39	IO13
GPORT4 BIT0	gpio44	A0
GPORT4 BIT1	gpio45	A1
GPORT4 BIT2	gpio46	A2
GPORT4 BIT3	gpio47	A3
GPORT4 BIT4	gpio48	A4
GPORT4 BIT5	gpio49	A5

**Tabela C.1.** Mapowanie portów Intel Galileo na pliki w systemie Linux

Źródło: <http://www.malinov.com/Home/sergey-s-blog>[12]

## DODATEK D

# Programy oraz dokumentacja

Katalog *Source* zawiera kod źródłowy oprogramowania stworzonego na potrzeby pracy wraz z dokumentacją

1. VM800Galileo.ino - główny kod programu
2. FT800.\* - obsługa wyświetlacza TFT przy użyciu protokołu komunikacyjnego SPI
3. FT800api.\* - API udostępniające podstawowe funkcje systemy HMI dostępne dla wyświetlacza TFT
4. I2C.\* - obsługa I/O Expander przy użyciu protokołu komunikacyjnego  $I^2C$
5. simulator.\* - komunikacja z symulatorem samochodu
6. doc - dokumentacja kodu wygenerowana za pomocą systemu *DOXYGEN*

# Bibliografia

- [1] Arduino. Arduino uno. <https://www.arduino.cc/en/Main/ArduinoBoardUno>. [Online; dostęp 12-03-2016].
- [2] Manoel Carlos Ramon. *Intel Galileo and Intel Galileo Gen2 API Features and Arduino projects for Linux programmers*. Apress Media, 2014.
- [3] Intel Corporation. Intel® galileo development board: Datasheet. <http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g1-datasheet.html>. [Online; dostęp 15-11-2015].
- [4] Arduino. Intel galileo. <https://www.arduino.cc/en/ArduinoCertified/IntelGalileo>. [Online; dostęp 15-11-2015].
- [5] FTDI. Ft800 - display, audio and touch controller. <http://www.ftdichip.com/Products/ICs/FT800.html>. [Online; dostęp 15-11-2015].
- [6] Byte Paradigm.  $i^2c$  vs spi. <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>. [Online; dostęp 15-11-2015].
- [7] Dorota Rabczuk. Szeregowy interfejs spi. <http://castor.am.gdynia.pl/~dorra/pliki/Magistrale%20%20-%20podstawy%20teoretyczne.pdf>. [Online; dostęp 15-11-2015].
- [8] JSON. Json - wprowadzenie. <http://www.json.org/json-pl.html>. [Online; dostęp 28-02-2016].
- [9] Bartłomiej Świercz. Wprowadzenie do technologii xml. [http://neo.dmc.s.pl/webservices/websry\\_wyklad1.pdf](http://neo.dmc.s.pl/webservices/websry_wyklad1.pdf). [Online; dostęp 28-02-2016].

- [10] Lynnette Reese. A comparison of open source hardware: Intel galileo vs. raspberry pi. <http://eu.mouser.com/applications/open-source-hardware-galileo-pi/>. [Online; dostęp 15-11-2015].
- [11] Botland. Arduino uno rev3. <http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html>. [Online; dostęp 15-11-2015].
- [12] Sergey Kiselev. Sergey's blog. <http://www.malinov.com/Home/sergey-s-blog>. [Online; dostęp 15-11-2015].

# **Spis tabel**

6.1. Porty użyte do działania systemu . . . . .	34
B.1. Specyfikacja dostępnych na rynku mikro kontrolerów . . . . .	48
C.1. Mapowanie portów Intel Galileo na pliki w systemie Linux . . .	49

# Spis rysunków

2.1.	Arduino Uno . . . . .	7
2.2.	Galileo Gen 1 Board . . . . .	9
2.3.	Schemat logiczny układu Intel Galileo . . . . .	10
2.4.	Architektura FTDI EVE VM800B . . . . .	11
2.5.	Bufor kołowy dostępny podczas programowania ekranu VM800	12
2.6.	Schemat połączenia wyświetlacza FTDI EVE z kontrolerem za pomocą SPI . . . . .	13
2.7.	Schemat połączenia symulatora samochodu z kontrolerom za pomocą $I^2C$ . . . . .	13
4.1.	Arduino Studio . . . . .	18
5.1.	Przebieg czasowy protokołu $I^2C$ . . . . .	24
5.2.	Schemat odbierania/wysyłania danych poprzez $I^2C$ na przykładzie PCF8574N . . . . .	26
5.3.	Przebiegi czasowe interfejsu SPI . . . . .	30
6.1.	Zdjęcie gotowego zestawu . . . . .	33
6.2.	Schemat logiczny zestawu . . . . .	34
6.3.	Schemat elektryczny symulatora samochodu . . . . .	36
6.4.	Schemat blokowy działania programu . . . . .	37
6.5.	Ekran startowy . . . . .	38
6.6.	Ekran główny . . . . .	38
6.7.	Ekran opcji . . . . .	39
6.8.	Strona www . . . . .	43

# 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