

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 jest stworzenie systemu komputera pokładowego do samochodu, w którego skład będzie wchodzić:

1. Mikrokomputer Intel Galileo Gen 1,
2. Wyświetlacz LCD FTDI EVE VM800B z panelem dotykowym,
3. Zestaw czujników symulujących odpowiedni dla rzeczywistego samochodu stan w szczególności:
 - (a) Informacje na temat aktualnej temperatury w samochodzie, na zewnątrz oraz w silniku
 - (b) Informacje na temat stanu drzwi (otwarте/zamknięte)
 - (c) Informacje na temat stanu pasów (zapięte/odpięte)
 - (d) Ekran z obrazem symulujący inteligentne lusterko wsteczne
4. Oprogramowanie

Słowa kluczowe

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

Spis treści

1. Wprowadzenie	5
1.1. Cele	5
1.2. Założenia	5
1.3. Plan pracy	6
2. Architektura	7
2.0.1. Intel Galileo	7
2.0.2. FTDI EVE VM800B	9
2.0.3. Symulator samochodu	12
3. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem	13
3.0.1. Odpytywanie w pętli	13
3.0.2. Porty	13
3.0.3. Przerwania	14
4. Programowanie z użyciem różnych środowisk	16
4.1. Programowanie w środowisku Arduino studio	16
4.2. Komunikacja z urządzeniami poprzez mechanizmy systemu operacyjnego	19
5. Implementacja	21
5.1. Protokół komunikacyjny I^2C	21
5.1.1. I/O Expander PCF8574N	22
5.2. Protokół komunikacyjny SPI	24
5.2.1. Komunikacja poprzez protokół SPI	26
6. Działanie komputera pokładowego	29
6.1. Założenia funkcjonalne projektu	29
6.2. Opis działania	29

6.3. Wnioski oraz własne doświadczenia	34
Zakończenie	35
A. Karty Katalogowe	36
B. Porównanie dostępnych na rynku mikro kontrolerów	37
C. Mapowanie portów Intel Galileo na pliki w systemie Linux	38
D. Programy	39
Bibliografia	40
Spis tabel	41
Spis rysunków	42
Oświadczenie	43

ROZDZIAŁ 1

Wprowadzenie

1.1. Cele

Celem pracy jest konstrukcja oraz oprogramowanie systemu komputera pokładowego do samochodu. Komputer ma wczytywać temperaturę z czujników panującą w silniku, na zewnątrz, w środku oraz aktualnym stanie zapięcia pasów i zamknięcia drzwi. Jednym z założeń tworzenia systemu jest konstrukcja modułowa umożliwiająca późniejszą rozbudowę funkcjonalności na przykład funkcję rejestrującą pozycję *GPS* wraz z zapisaniem na karcie pamięci *microSD*. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie wyświetlacza LCD *FTDI EVE VM800B* z panelem dotykowym.

1.2. Założenia

Do wykonania pracy zostały przyjęte następujące założenia:

1. Użycie mikrokomputera *Intel Galileo Gen 1* jako głównego silnika dla całego komputera wraz z zainstalowanym systemem operacyjnym *Linux YOCTO*
2. Wyświetlacz LCD *FTDI EVE VM800B* z panelem dotykowym jako interfejs komunikacyjny komputera z użytkownikiem,
3. Kompilacja przy użyciu systemu Arduino studio oraz natywnego systemu dla Galileo - Linux YOCTO
4. Symulacja funkcjonalności rzeczywistych czujników samochodu za pomocą dedykowanego symulatora składającego się z zestawu przełączni-

ków zapięcia pasów/zamknięcia drzwi oraz potencjometrów służących jako analogowe czujniki 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

TO DO

ROZDZIAŁ 2

Architektura

2.0.1. Intel Galileo

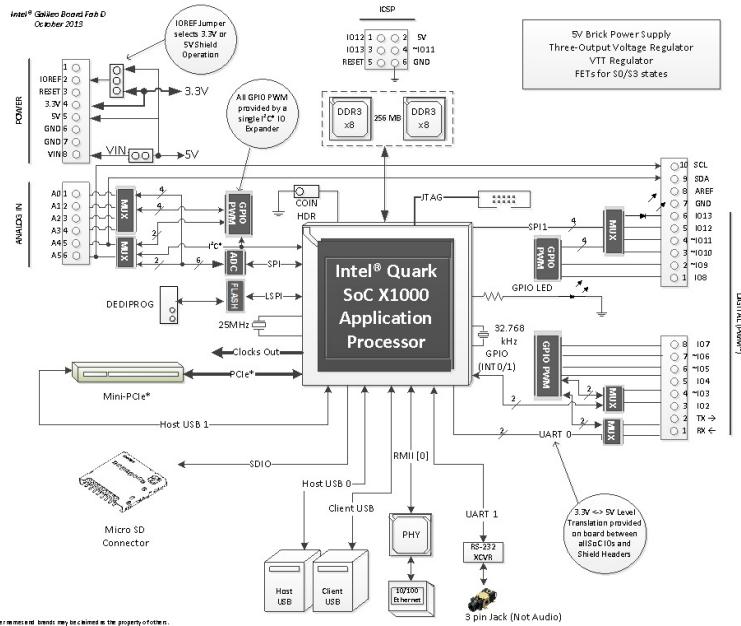
Intel Galileo jest to mikro kontroler oparty na 32-bitowym procesorze Intel® Quark SoC X1000 i taktowaniu 400MHz. Został on wyposażony w standar-dowy interfejs Arduino składający się z: 14 pinów cyfrowych (w tym 6 pinów mogących pełnić funkcję *PWM*¹) oraz 6 pinów analogowych. Każdy z tych pinów jest w stanie operować napięciem od 0V do max 5V. Bardzo dużym atutem Galileo jest wbudowana karta sieciowa, port *RS-232*, port *USB* oraz slot karty *microSD*.[1]



Rysunek 2.1. Galileo Gen 1 Board

Źródło: <http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g1-datasheet.html>[2]

¹ang. Pulse-Width Modulation - technika pozyskiwania wyników analogowych poprzez użycie wyjść cyfrowych



Rysunek 2.2. Schemat logiczny układu Intel Galileo

Źródło: [https://www.arduino.cc/en/ArduinoCertified/IntelGalileo\[3\]](https://www.arduino.cc/en/ArduinoCertified/IntelGalileo[3])

Do komunikacji z Galileo programista ma do dyspozycji port RS-232, USB (działające w trybie host oraz client) oraz wyjście Ethernet. Intel Galileo jest zasilane napięciem 5V i 2.0A, które może zostać dostarczone poprzez zasilacz z zestawu lub poprzez podłączenie zasilania do portów PWR² oraz GND³. Standardowo środowiskiem programistycznym jest Arduino Studio. Programista pisząc w języku C i używając dostarczonych przez producenta Arduino funkcji do obsługi portów może się z nimi komunikować. Następnie przesyła skompilowaną wersję poprzez kabel USB do urządzenia. Po przesłaniu program zostaje automatycznie załadowany do pamięci urządzenia i uruchomiony. Gdy mamy zainstalowany system operacyjny wtedy komunikację

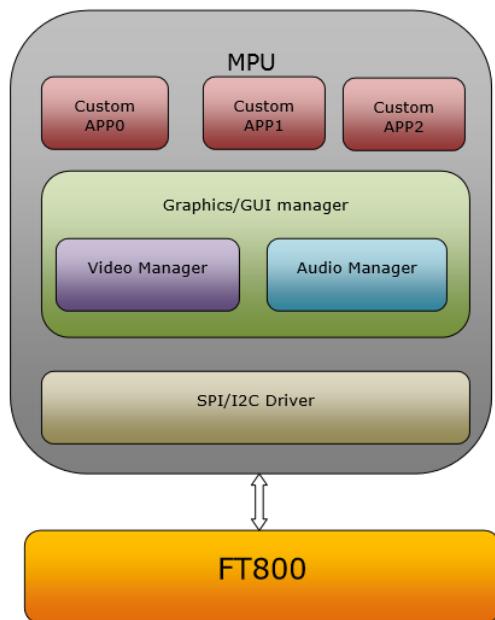
²Port używany jako port zasilania (5V)

³Port używany jako masa - GROUND

można prowadzić w dowolnym języku programowania łącząc się z systemem poprzez SSH⁴ lub port RS-232.

2.0.2. FDTI EVE VM800B

Do komunikacji komputera z użytkownikiem został użyty wyświetlacz *LCD FDTI EVE VM800B* wraz z panelem dotykowym oraz wbudowanym kontrolerem audio.



Rysunek 2.3. Architektura Ekranu FDTI EVE VM800B

Źródło: FT800 Programmers Guide

Podstawowe cechy urządzenia[6]:

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

⁴ang. secure shell - protokół komunikacyjny służący do połączenia się ze zdalnym komputerem będącym w sieci

3. 262 tys. kolorów
4. Możliwość wygładzania krawędzi
5. Możliwość komunikacji poprzez użycie interfejsu I^2C lub SPI
6. Wbudowany system HMI⁵
7. Zakres pracy wyświetlacza: $-40^\circ C$ do $85^\circ C$

Komunikacja Galileo z Ekranem odbywa się poprzez protokół komunikacyjny *SPI*. Do tego celu została napisana własna wersja funkcji służących do wysłania oraz odczytania danych z ekranu (dokładny opis w rozdziale: "Implementacja").

Rozpoczęcie pracy wyświetlacza polega na wpisaniu określonych wartości do pewnych obszarów jego pamięci określając w ten sposób np. rozdzielcość lubłączenie/wyłącznie modułu odpowiedzialnego za dźwięk czy dotyk.

Poprzez pamięć urządzenia możemy rozumieć tablicę, w której każda komórka ma swój adres. Wysłanie pewnej wartości do pamięci oznacza więc wpisanie tej wartości do określonej przez adres komórki w tablicy.

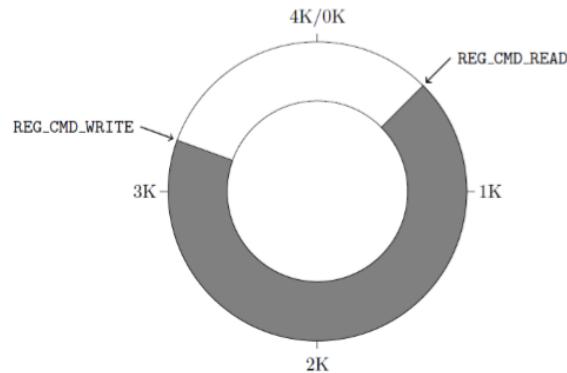
Kolejnym krokiem jest wywołanie zestawu funkcji odpowiedzialnych za rysowanie niezbędnych do działania systemu elementów np. guzików. Wszystkie wyświetlane elementy są na początku zapisywane w buforze pamięci. Następnie gdy cały ekran zostanie już przygotowany następuje wyświetlenie wszystkiego co zostało zapisane do bufora po czym zostaje on wyczyszczony.

Procedura rysowania wygląda następująco:

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

⁵ang. Human - Machine Interface - system bazujący na widgetach

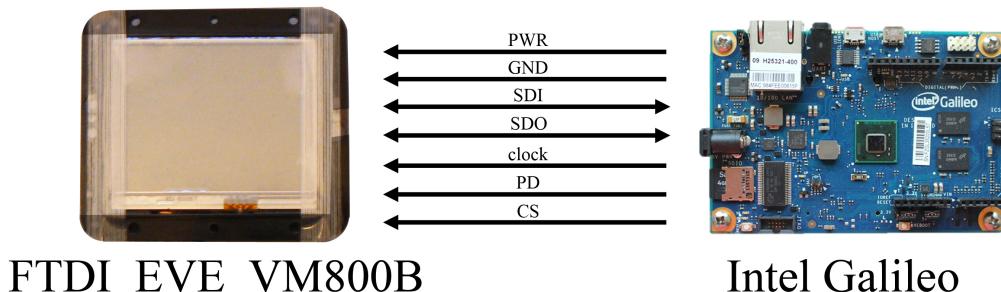
5. Wyświetl wszystko do zostało dodane do bufora



Rysunek 2.4. Bufor dostępny podczas programowania ekranu VM800

Źródło: FT800 Programmers Guide

Należy pamiętać, że wielkość bufora, którą mamy do dyspozycji wynosi 4 Kb.



Rysunek 2.5. Schemat połączenia wyświetlacza FTDI EVE z Intel Galileo za pomocą SPI

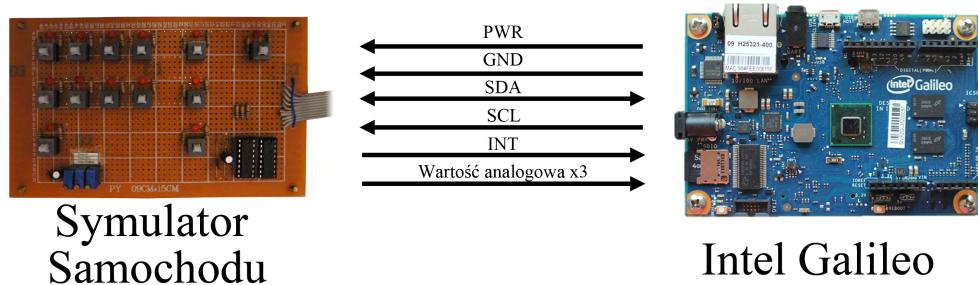
Źródło: Opracowanie własne

2.0.3. Symulator samochodu

Do celów demonstracyjnych oraz implementacjach komputer nie został zamontowany w fizycznym samochodzie. Z tego powodu został zabudowany symulator samochodu mający obrazować pełną pracę pojazdu składający się z:

1. 11 przełączników bistabilnych symulujących drzwi samochodu (wraz z bagażnikiem), pasy pasażerów, wrzucenie biegu wstecznego oraz włączenie świateł - wcisnięcie przełącznika obrazowane jest poprzez zaświecenie diody półprzewodnikowej
2. 2 I/O Expander PCF 8574N
3. 3 potencjometrów symulujących czujniki temperatury w samochodzie

Przełączniki wysyłają sygnały cyfrowe, a potencjometry sygnały analogowe. Ze względu na zbyt dużą ilość sygnałów cyfrowych wychodzących z symulatora komunikacja z Galileo odbywać się będzie poprzez *I/O Expander PCF 8574N* wykorzystujący protokół I^2C . Do obsługi tego protokołu również została napisana własna wersja oprogramowania (dokładny opis w rozdziale: "Implementacja").



Rysunek 2.6. Schemat połączenia symulatora samochodu z Intel Galileo za pomocą I^2C

Źródło: Opracowanie własne

ROZDZIAŁ 3

Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem

3.0.1. 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 cały czas zajmuje niepotrzebnie zasoby sprzętu niepotrzebnymi zapytaniami. Odczyt stanu urządzeń wejścia/wyjścia może być realizowane mechanizmami systemu operacyjnego lub niskopoziomowo z wykorzystaniem funkcji języka C i/lub Asemblera.

3.0.2. 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). Najczęściej wysłanie sygnału równego 1 jest równoznaczne z wysłaniem napięcia o wartości 5V oraz odpowiednio wysłanie 0 jest równoznaczne z wysłaniem napięcia równego 0V. Z kolei porty analogowe mogą przesyłać sygnały nawet 10 bitowe. Każdy z portów może działać w jednym z dwóch trybów: wejścia - oczekiwany na przyjęcie danych od urządzenia zewnętrznego oraz wyjścia - wysyłać dane do urządzenia zewnętrznego.

3.0.3. Przerwania

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

Są to bezpośrednie funkcje systemu lub sprzętu ułatwiające komunikację ze światem zewnętrznym. Część z nich jest zarezerwowana przez system lecz część jest wolna do wykorzystania przez programistę.

System Galileo bazując na procesorze Quark ma standardowy dla produktów Intela mechanizm obsługi przerwań obsługiwany za pomocą kontrolera przerwań więc do dyspozycji mamy przerwania:

1. Programowe
2. Sprzętowe
 - (a) Niemaskowalne (NMI¹)
 - (b) Maskowalne
3. Wyjątek

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

Przerwania programowe wywołuje się za pomocą komendy *INT XX*, gdzie *XX* oznacza numer przerwania zadeklarowanego w tablicy wektorów przerwań², 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 zgłasiane przez procesor. Zostają one wywołane niezależnie w określonych przypadkach. Przerwania te dzielimy na maskowalne oraz niemaskowalne. Główna różnica między nimi polega na możliwości

¹Non-Maskable Interrupt

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

zablokowania przerwań maskowalnych podczas gdy przerwania niemaskowalne muszą zostać obsłużone. Przykładem przerwania niemaskowalnego jest *INT2* czyli popularny *Blue Screen of Death*³.

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

Mechanizmem, który bazuje na przerwaniach jest mechanizm Timera. Polega on na wywołaniu funkcji co określony czas (zgłaszone jako przerwanie), którego zarządzaniem zajmuje się urządzenie (lub system operacyjny). Rozwiązanie to jest bardzo podobne do odpytywania w pętli, a następnie wywołania funkcji *delay()* z tą różnicą, że użycie timera jest dokładniejsze ponieważ wykorzystuje zegar czasu rzeczywistego znajdującego się w CPU⁴.

³Ekran błędu w systemach Windows pojawiający się po krytycznym błędzie systemu

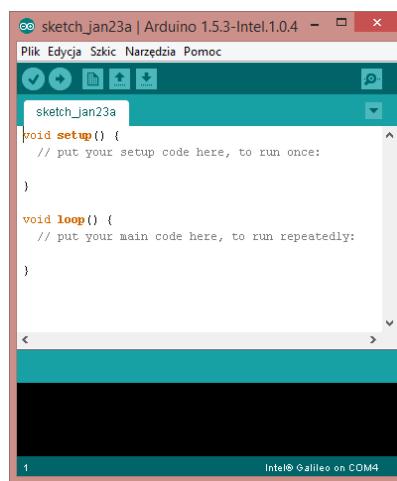
⁴ang. Central Processing Unit - jednostka arytmetyczno-logiczna wykorzystywana do wykonywania obliczeń niezbędnych do działania programu

ROZDZIAŁ 4

Programowanie z użyciem różnych środowisk

4.1. Programowanie w środowisku Arduino studio

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



Rysunek 4.1. Arduino Studio

Źródło: Opracowanie własne

Po skompilowaniu i przesłaniu programu z Arduino Studio do Galileo jednorazowo uruchomiona zostaje funkcja *setup()*, 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. Taką sytuację można utożsamiać z wywołaniem jakiejkolwiek funkcji w bloku *while(1)*.

Listing 4.1. Odpytywanie w nieskończonej pętli w środowisku Arduino

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

W środowisku *Arduino* aby obsłużyć port analogowy wystarczy ustalić tryb w jakim ma on działać (wejście/wyjście), a następnie wysłać/odczytać dane. Wpisanie wartości *LOW* jest równoznaczna z wysyłaniem napięcia 0V na określonym pinie, a wpisanie wartości *HIGH* jest równoznaczna z wysyłaniem napięcia 5V na określonym pinie. Arduino dostarcza funkcje 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)
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 4.2. Obsługa portu cyfrowego w środowisku Arduino

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

Listing 4.3. Obsługa portu analogowego w środowisku Arduino

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

Arduino oczywiście obsługuje przerwania. Ich obsługa jest bardzo prosta. W środowisku Arduino aby obsłużyć port analogowy wystarczy wywołać funkcję *attachInterrupt*:

Listing 4.4. 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, 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

4.2. Komunikacja z urządzeniami poprzez mechanizmy systemu operacyjnego

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

Obsługa Timera jest analogiczna do obsługi przerwań.

Listing 4.5. Przykładowe użycie timer w środowisku Arduino

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

4.2. Komunikacja z urządzeniami poprzez mechanizmy systemu operacyjnego

Z punktu widzenia systemu operacyjnego urządzenia 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 tzn. 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` trzeba wpisać `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 - odpowiednik `HIGH` z Arduino lub `0` dla ustawienia stanu niskiego - odpowiednik `LOW` z Arduino, gdzie `PORT` jest to numer odpowiedniego portu według numeracji Galileo. Warto zauważyć, że przesyłane wartości nie są wartościami 8 tylko 1 bitowymi.

Dla języka C wygląda to następująco:

Listing 4.6. Obsługa portu cyfrowego w środowisku Linux (język 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 wartości do portu cyfrowego
fp = fopen( "/sys/class/gpio/gpio39/value" , "w" );
fprintf( fp , "1" );
fclose( fp );

// Odczytanie wartości 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 4.7. Obsługa portu cyfrowego w środowisku Linux (bash)

```

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

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

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

```

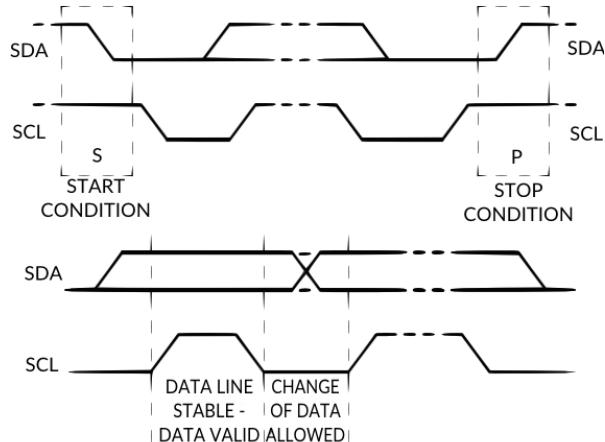
ROZDZIAŁ 5

Implementacja

5.1. Protokół komunikacyjny I^2C

I^2C ¹ jest szeregowym interfejsem stworzonym przez firmę *Philips* służącym do przesyłania danych między urządzeniami elektrycznymi.

Podstawową cechą I^2C jest wykorzystywanie dwóch linii służących do komunikacji: dwukierunkowa linia danych SDA ² oraz jednokierunkowa linia zegarowa SCL ³. Protokół I²C bazuje na przesyłaniu ramek (pakietów) składających się z sekwencji: START -> dane -> STOP.



Rysunek 5.1. Przebieg czasowy protokołu I^2C

Źródło: <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>^[4]

¹ang. Inter-Integrated Circuit

²ang. Serial Data Line

³ang. Serial Clock Time

Do wygenerowania impulsu START oraz STOP wystarczy ustawić linię *SDA* oraz *SCL* w stan HIGH (5V) po czym należy chwilkę odczekać. Dane wysyłane/odbierane są bit po bicie - na początku należy ustawić linię *SCL* w stan wysoki (HIGH), odczytać wartość na porcie *SDA*, a następnie zmienić stan linii *SCL* na niski (LOW).

Dane wysyłane są od najstarszego do najmłodszego bitu. Każda paczka potwierdzona jest przez odbiornik (bit ACK⁴). Należy również pamiętać, aby każdą komunikację z urządzeniem rozpoczęć i zakończyć ustawiając linie *SDA* oraz *SCL* w stan nieaktywny (HIGH) - wygenerowanie sekwencji STOP.

Podstawowymi zaletami protokołu są:

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

Nazwa *I²C* jest nazwą zastrzeżoną dlatego też w literaturze bardzo często spotyka się określenie *TWI*⁵. Jest ono stosowane w mikro kontrolerach firmy *Atmel*.

5.1.1. I/O Expander PCF8574N

W symulatorze ze względu na dużą ilość wychodzących sygnałów zastosowano komunikacje poprzez *I²C*. Do obsługi tego został zamontowane dwa I/O Expander PCF 8574N zbierające wszystkie sygnały cyfrowe wychodzące z symulatora do Galileo. Dzięki temu zamiast używać dwunastu linii cyfrowych, wykorzystywane są jedynie dwie niezbędne do komunikacji poprzez *I²C* co znacznie ułatwiło dalsze korzystanie z Galileo ze względu na pozostałe wolne porty cyfrowe, które będą potrzebne w dalszej części do komunikacji

⁴ang. Acknowledge

⁵ang. Two Wire Interface

z wyświetlaczem. Zaletą tego rozwiązania jest możliwość późniejszego podłączenia większej ilości czujników bez jakiegokolwiek ingerencji w okablowanie Galileo.

Do rozpoczęcia komunikacji należy wygenerować bit START oraz zaadresować urządzenie, z którym będziemy się komunikować.

Adresowanie urządzenia odbywa się poprzez wysłanie pojedynczych bitów adresu (pamiętając o kolejności MSB->LCB⁶) oraz wygenerowanie impulsu zegara. Gdy chcemy zaadresować urządzenie, którego adresem jest np. 4 należy wykonać:

Listing 5.1. Adresowanie urządzenia I^2C na przykładzie PCF8574N

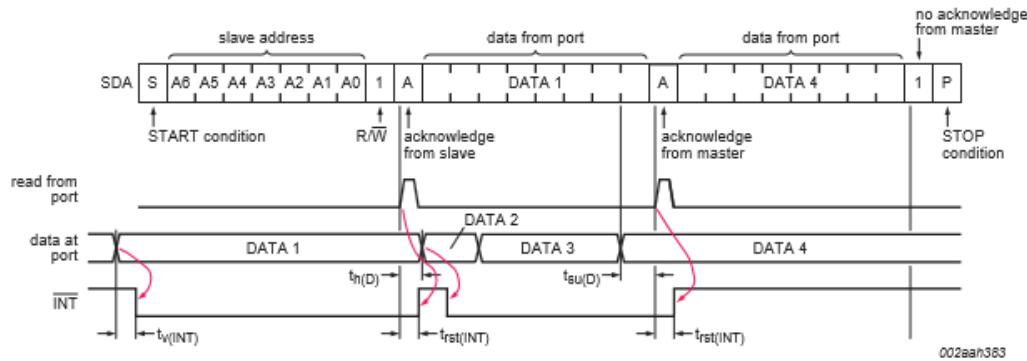
```
int adres = 4;
for (m = 0x80; m; m >>= 1){
    if (adres & m)
        digitalWrite(sda, HIGH);
    else
        digitalWrite(sda, LOW);

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

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

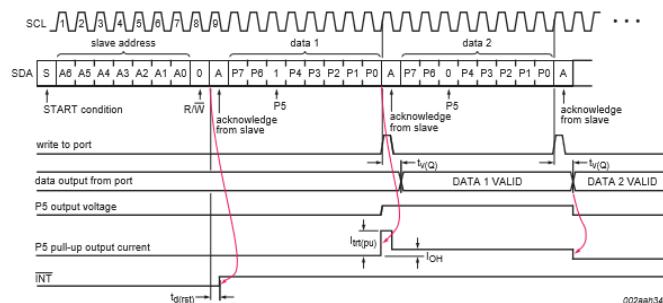
Galileo posiada dostarczoną od Arduino bibliotekę do obsługi I^2C jednak podczas próby użycia jej w projekcie wystąpiły problemy z kompatybilnością z używanym sprzętem w związku z tym na potrzeby projektu napisana została własna wersja biblioteki obsługującej komunikację poprzez protokół I^2C .

⁶Wysyłanie odbywa się w kolejności od najbardziej znaczących (najstarszych) bitów



Rysunek 5.2. Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N



Rysunek 5.3. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

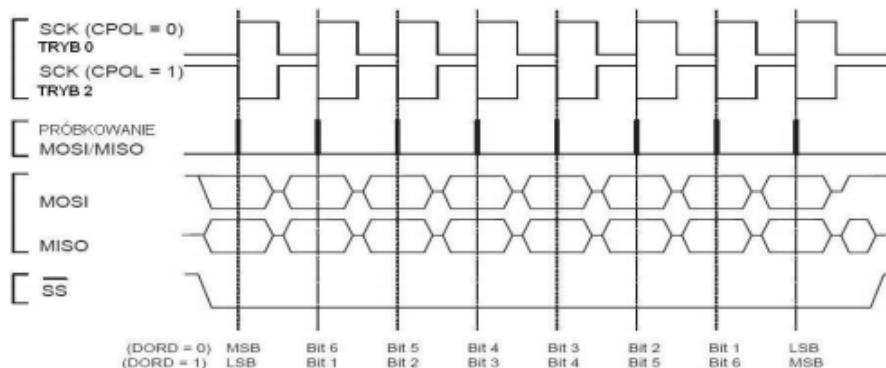
5.2. Protokół komunikacyjny SPI

Wyświetlacz LCD wyposażony jest w protokół SPI⁷. Składa się on z czterech podstawowych linii - dwóch służących do przesyłania danych w przeciwnych

⁷ang. Serial Peripheral Interface

kierunkach, jednej z sygnałem taktującym, synchronizującym transfer danych oraz linii *Chip Select*.

Linia MISO⁸ jest linią wejścia danych dla urządzenia nadzawanego (master), a wyjściem dla urządzenia podrzędnego (slave), linia MOSI⁹ jest wyjściem dla urządzenia master, a wejściem dla slave. Linia SCK¹⁰ 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.[5]



Rysunek 5.4. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0

Źródło: <http://castor.am.gdynia.pl/~dorra>[5]

⁸ang. Master In Slave Out

⁹ang. Master Out Slave In

¹⁰ang. Serial Clock

5.2.1. Komunikacja poprzez protokół SPI

Komunikacja zawsze przebiega dwustronnie (duplex¹¹). 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. W porównaniu z I^2C protokół SPI jest dużo szybszy. Maksymalna prędkość protokołu I^2C w wydaniu z 2012r. to 5-MHz.

Podobnie jak w przypadku protokołu I^2C została napisana własna wersja biblioteki do obsługi protokołu SPI. Do komunikacji niezbędne było napisanie funkcji:

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 ($FT800_{ACTIVE}$)

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

1. inicjalizacja ekranu
2. rysowanie kółka o podanym rozmiarze w podanym miejscu
3. rysowanie linii po podanych końcach

¹¹Nadawanie i odbieranie informacji odbywa się w obu kierunkach

4. wypisanie tekstu
5. wypisanie cyfr
6. narysowanie guzika po podanym miejscu i podanym rozmiarze

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

Listing 5.2. 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 5.3. Narysowanie kropki na ekranie

```
void kropka(unsigned long color, unsigned int size, unsigned long x,
            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 była komunikacja z zestawem czujników, które mogą być zamontowane w samochodzie.

W projekcie zostały użyte czujniki otwarcia/zamknięcia drzwi, zapięcia pasów,łączenia/wyłączenia światel oraz czujniki temperatury. Dodatkowym elementem była komunikacja z zewnętrznym wyświetlaczem LCD służącym do komunikacji pomiędzy użytkownikiem a komputerem. Samochód obrazowany został za pomocą zbudowanego symulatora.

6.2. Opis działania

Proponowany komputer pokładowy jest całkowicie osobnym systemem niezależnym od sprzętu aktualnie posiadanego w samochodzie. Użytkownik montuje zestaw czujników oraz łączy je z komputerem i ekranem. Po wejściu do samochodu oraz włączeniu zapłonu powoduje automatyczny start systemu.

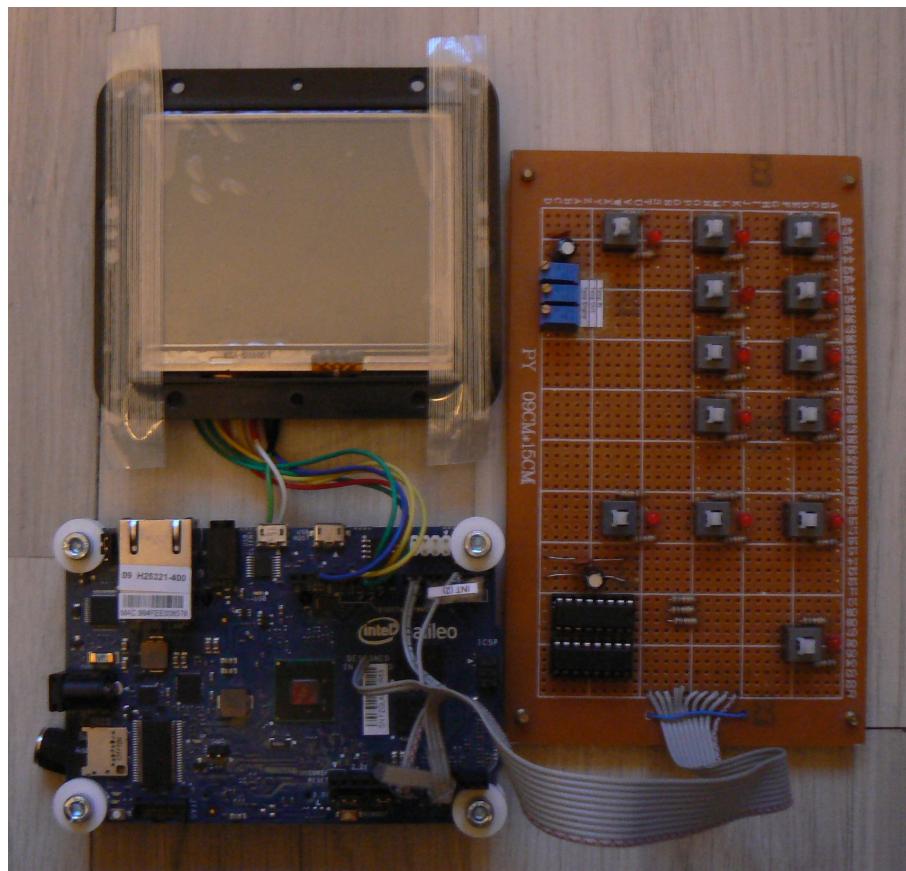
Na ekranie pojawia się powitanie oraz ekran startowy, na którym można zobaczyć symulację aktualnej pozycji GPS, temperaturę panującą w środku samochodu, na zewnątrz oraz w silniku. Stan drzwi i pasów z wizualizowany został poprzez miniaturkę samochodu z aktywnie otwierającymi się drzwiami, zapalającymi światłami oraz ikonką obrazującą stan zapiętych pasów.

Na ekranie widnieją 2 przyciski - *Save data* oraz *Smart Mirror*. Pierwszy z nich daje możliwość zapisu aktualnej pozycji GPS oraz temperatur na karcie pamięci microSD, drugi przechodzi w tryb aktywnego lusterka wstecznego, który może również służyć jako czujnik cofania co bardzo przydaje się podczas parkowania w ciasnych miejskich parkingach. Wyłączenie systemu

następuje wraz z wyłączeniem zapłonu w samochodzie. Zapis do pliku odbywa się poprzez użycie standardowych funkcji systemu operacyjnego.

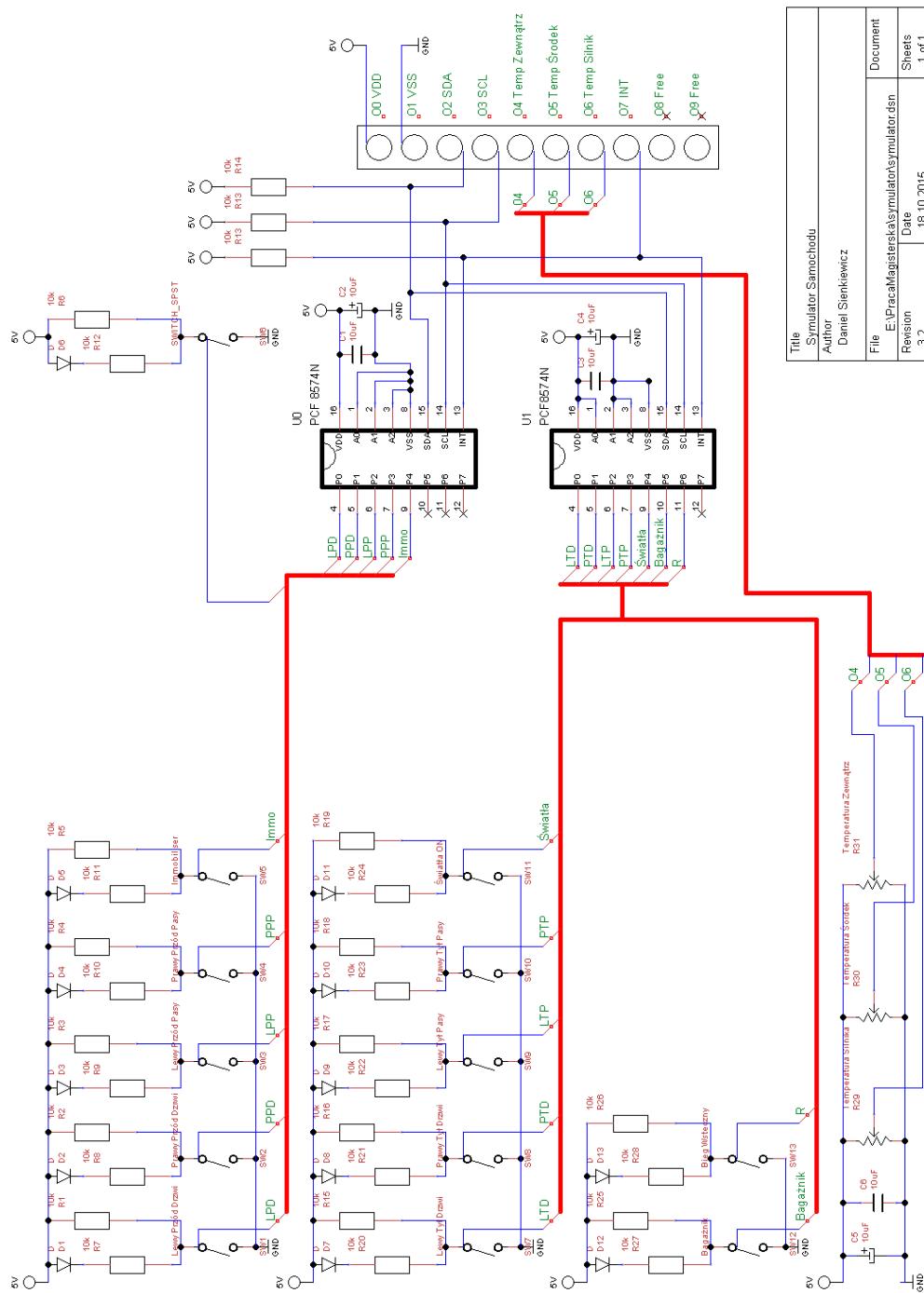
Na początku przygotowana zostaje komenda a następnie poprzez wywołanie funkcji *system()* zostaje ona wykonana. Do tego celu można również użyć standardowego mechanizmu dostępnego w Arduino. Za pomocą funkcji *SD.open(filename)* należy otworzyć plik na karcie pamięci po czym zapis polega na wpisaniu danych za pomocą funkcji *println()*.

W tym przypadku należy pamiętać o zainportowaniu pliku nagłówkowego *SD.h*.



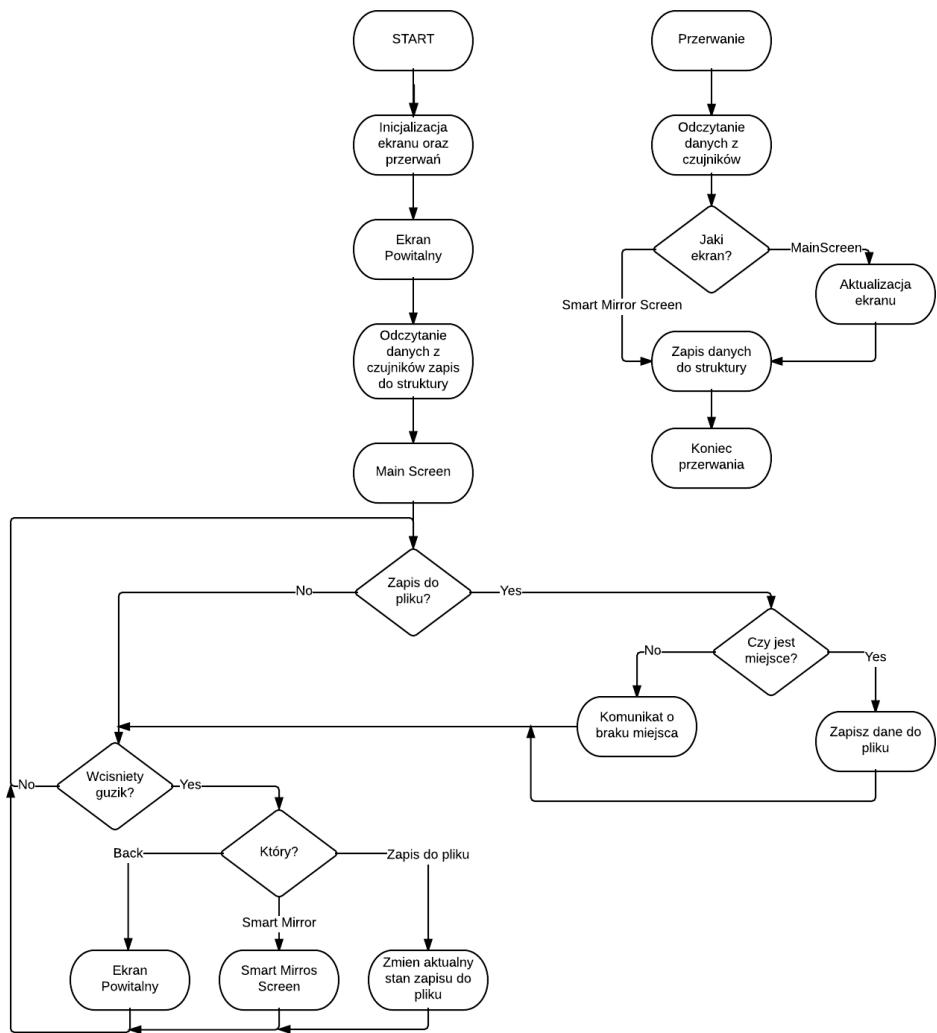
Rysunek 6.1. Zdjęcie gotowego zestawu

Źródło: Opracowanie własne



Rysunek 6.2. Schemat elektryczny symulatora samochodu

Źródło: Opracowanie własne



Rysunek 6.3. Schemat blokowy programu

Źródło: Opracowanie własne

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

```

String command = "";
command = "echo testTekst";
command += ">>/tmp/daniel.txt";
system(command.buffer);

```

Komunikacja z Intel Galileo z ekranem odbywa się używając protokołu SPI. Łącznie użytych zostało 7 linii: SDI, SDO, clock, PD, CS¹ oraz PWR² i GND³.

Komunikacja komputera z symulatorem odbywa się poprzez użycie I^2C . Odczytanie wartości czujników cyfrowych odbywa się po otrzymaniu przerwania sprzętowego wychodzącego z I/O Expander przy zmianie jakiejkolwiek wartości np. przy otworzeniu drzwi.

¹Linia Chip select wyświetlacza

²ang. Power - 5V

³ang. Ground

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 aktualna pozycja GPS zostaje zapisywana do pliku tekstuowego do momentu wyłączenia systemu, wyczerpania miejsca na karcie lub zmiany tej opcji przez użytkownika.

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.

Zakończenie

TO DO

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	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/> [7]

Źródło: <http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html> [8]

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>[9]

DODATEK D

Programy

Katalogi *Source* zawiera kod źródłowy oprogramowania stworzonego na potrzeby pracy.

1. VM800Galileo.ino - główny kod programu
2. FT800.* - obsługa wyświetlacza LCD przy użyciu protokołu komunikacyjnego SPI
3. FT800api.* - API udostępniające podstawowe funkcje systemy HMI dostępne dla wyświetlacza LCD
4. I2C.* - obsługa I/O Expander przy użyciu protokołu komunikacyjnego I^2C
5. simulator.* - komunikacja z symulatorem samochodu

Bibliografia

- [1] Manoel Carlos Ramon. *Intel Galileo and Intel Galileo Gen2 API Features and Arduino projects for Linux programmers.* Apress Media, 2014.
- [2] 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].
- [3] Arduino. Intel galileo. <https://www.arduino.cc/en/ArduinoCertified/IntelGalileo>. [Online; dostęp 15-11-2015].
- [4] Byte Paradigm. i^2c vs spi. <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>. [Online; dostęp 15-11-2015].
- [5] Dorota Rabczuk. Szeregowy interfejs spi. <http://castor.am.gdynia.pl/~dorra/pliki/Magistrale%20%20-%20podstawy%20teoretyczne.pdf>. [Online; dostęp 15-11-2015].
- [6] FTDI. Ft800 - display, audio and touch controller. <http://www.ftdichip.com/Products/ICs/FT800.html>. [Online; dostęp 15-11-2015].
- [7] 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].
- [8] Botland. Arduino uno rev3. <http://botland.com.pl/arduino-moduly-glowne/1060-arduino-uno-r3.html>. [Online; dostęp 15-11-2015].
- [9] Sergey Kiselev. Sergey's blog. <http://www.malinov.com/Home/sergey-s-blog>. [Online; dostęp 15-11-2015].

Spis tabel

B.1. Specyfikacja dostępnych na rynku mikro kontrolerów	37
C.1. Mapowanie portów Intel Galileo na pliki w systemie Linux . . .	38

Spis rysunków

2.1.	Galileo Gen 1 Board	7
2.2.	Schemat logiczny układu Intel Galileo	8
2.3.	Architektura Ekranu FTDI EVE VM800B	9
2.4.	Bufor dostępny podczas programowania ekranu VM800	11
2.5.	Schemat połączenia wyświetlacza FTDI EVE z Intel Galileo za pomocą SPI	11
2.6.	Schemat połączenia symulatora samochodu z Intel Galileo za pomocą I^2C	12
4.1.	Arduino Studio	16
5.1.	Przebieg czasowy protokołu I^2C	21
5.2.	Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N	24
5.3.	Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N	24
5.4.	Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0	25
6.1.	Zdjęcie gotowego zestawu	30
6.2.	Schemat elektryczny symulatora samochodu	31
6.3.	Schemat blokowy programu	32

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