UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Daniel Sienkiewicz
nr albumu: 206358

Projekt komputera samochodowego bazujący na systemie mikrokomputera Intel Galileo

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

Gdańsk 2015

Streszczenie

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

1. Mikrokomputer Intel Galileo Gen 1,
2. Ekran dotykowy FTDI EVE VM800B,
3. Oprogramowanie,

Słowa kluczowe

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

Spis treści

1. Wprowadzenie 5
1.1. Cele 5
1.2. Założenia 5
1.3. Plan pracy 6

2. Architektura 7
2.0.1. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem 7

3. Implementacja 13
3.0.1. Intel Galileo 13
3.1. Protokół komunikacyjny I^2C 15
3.1.1. Użycie protokołu I^2C na przykładzie I/O Expander PCF8574N 16
3.1.2. Problemy z bibliotekami 18
3.1.3. Własna implementacja I^2C 18
3.1.4. Własna biblioteka do komunikacji poprzez I^2C dla Intel Galileo 18
3.2. Protokół komunikacyjny SPI 18
3.2.1. Komunikacja poprzez protokół SPI 19
3.2.2. VM800B 20
3.3. Założenia funkcjonalne projektu 20
3.3.1. Symulator samochodu 20
3.4. Dalsze kroki oraz propozycje 21

Zakończenie 23

A. Karty Katalogowe 24

B. Porównanie dostępnych na rynku mikro kontrolerów 25

C. Mapowanie portów Intel Galileo na pliki w systemie Linux 26

D. Programy 27

Spis tabel 28

Spis rysunków 29

Oświadczenie 30

ROZDZIAŁ 1

Wprowadzenie

1.1. Cele

Celem pracy jest budowa oraz oprogramowanie komputera pokładowego do samochodu. Komputer powinien móc wczytać z czujników temperaturę panującą w silniku, na zewnątrz oraz w środku samochodu. Ponadto powinien on móc zapisać aktualną pozycję *GPS* na karcie pamięci microSD oraz umożliwić korzystanie z kamerki cofania lub inteligentnego lusterka wstecznego. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie ekranu dotykowego *FTDI EVE VM800B*.

1.2. Założenia

Do wykonania komputera wykorzystano: *Intel Galileo GEN 1* wraz z zainstalowanym oprogramowaniem *Linux YOCTO*, *Arduino IDE*, lokalizator *GPS* służący do podawania aktualnej pozycji dzięki której obliczana zostaje droga przebyta przez samochód, kamerka internetowa służąca jako czujnik cofania oraz inteligentne lusterko wsteczne oraz symulator samochodu. Aktualnie komputer nie będzie zamontowany do fizycznego samochodu więc do tych celów zbudowany został symulator składający się z podstawowych czujników takich jak: guziki służące za czujnik zapięcia pasów/zamknięcia drzwi, potencjometry służące za czujniki temperatury oraz *I/O expander PCF8574N* pozwalający na komunikację z *Intel Galileo*. Na komputerze nie będzie wyświetlana aktualna prędkość ani przebieg ponieważ nawet w najnowszych samochodach nie jest to dostępna opcja. Dane te są dostępne na zegarach samochodowych więc nie ma potrzeby powtarzania tej informacji.

1.3. Plan pracy

Pierwszy rozdział opisuje podstawowe cele oraz założenia projektu.

Druga część pracy przedstawia architekturę projektu wraz z jego opisem funkcjonalnym. Opisuję również mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem. Do komunikacji pomiędzy urządzeniami zostały wykorzystane 2 podstawowe protokoły komunikacyjne: I^2C oraz *SPI*.

W następnym rozdziale przedstawiony zostaje pomysł implementacji oraz proces tworzenia niezbędnej do obsługi symulatora samochodu biblioteki pozwalającej na komunikację poprzez *I/O Expander PCF8574N* z *Intel Galielo*. Zostaje tutaj również opisany proces tworzenia oprogramowania ekranu dotykowego *FTDI EVE VM800B* wraz z biblioteką niezbędą do komunikacji pomiędzy Galileo i ekranem.

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

ROZDZIAŁ 2

Architektura

2.0.1. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem

2.0.1.1. Porty

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

1. Cyfrowe
2. Analogowe

Porty cyfrowe charakteryzują się możliwością przyjęcia lub wysłania sygnału binarnego (1 - jest sygnał, 0 - sygnału nie ma). 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 - oczekiwać na przyjęcie danych od urządzenia zewnętrznego oraz wyjścia - wysyłać dane do urządzenia zewnętrznego.

W środowisku Arduino aby obsłużyć port analogowy wystarczy ustaliś tryb w jakim ma on działać (wejście/wyjście) a następnie wysłąć/odczytać dane. Dla portu analogowego wygląda to następująco:

Listing 2.1. Obsługa portu analogowego w środowisku Arduino

```
int val = 0;
int analogPin = A1;
pinMode(analogPin, OUTPUT);
```

```
val = analogRead(analogPin);
pinMode(analogPin, INPUT);
analogWrite(ledPin, val);
```

oraz odpowiednio dla portu cyfrowego:

Listing 2.2. Obsługa portu cyfrowego w środowisku Arduino

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

W środowisku linux każdy z portów możemy traktować jako plik, więc wpisanie lub odczytanie danych z portu jest równoznaczne z odpowiednio wpisaniem lub odczytaniem danych z pliku. Komunikację z portem możemy prowadzić w dowolnym języku. Dla języka C wygląda to następująco:

Listing 2.3. Obsługa portu cyfrowego w środowisku Linux (język C)

```
FILE *fp;
int value;

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

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

```
// Odczytanie świartoci 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 2.4. Obsługa portu cyfrowego w środowisku Linux (bash)

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

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

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

Przy komunikacji należy jednak pamiętać, że w Galileo nazwy urządzeń nie są intuicyjne tzn. port IO4 nie jest plikiem /sys/class/gpio/gpio4.

2.0.1.2. Przerwania

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

1. Programowe
2. Sprzętowe
   (a) Niemaskowalne (NMI[1])

[1]Non-Maskable Interrupt

(b) Maskowalne

3. Wyjątek

Przerwania programowe wywołuje się za pomocą komendy *INT XX*, gdzie *XX* oznacza numer przerwania zadeklarowanego w tablicy wektorów przerwań[2], która jest tworzona przy każdorazowym starcie systemu. Przerwanie to może przyjąć wartości do 255 i są one zarezerwowane przez procesor oraz użytkownika.

Przerwanie sprzętowe jest to rodzaj przerwań wywoływanych przez urządzenia wejścia/wyjścia lub zgłaszane przez procesor. Zostają one wywołane niezależnie w określonych przypadkach. Przerwania te dzielimy na maskowalne oraz niemaskowalne. Główna różnica między nimi polega na możliwości zablokowania przerwań maskowalnych podczas gdy przerwania niemaskowalne muszą zostać obsłużone. Przykładem przerwania niemaskowalnego jest *INT2* czyli popularny *Blue Screen of Death*[3].

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

Listing 2.5. 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. *Mode* może przyjmować wartości:

1. LOW - przerwanie zostanie zgłoszone gdy wartość na określonym pinie jest równa LOW[4]

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

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

[4]Wartość w środowisku arduino równoznaczna z napięciem 0V na określonym pinie

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[5]
4. FALLING - przerwanie zostanie zgłoszone gdy wartość na określonym pinie zostanie zmieniona z HIGH na LOW

2.0.1.3. Odpytywanie w pętli

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

Listing 2.6. Odpytywanie w nieskończonej pętli w środowisku Arduino

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

Funkcja *loop()* 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 sobie z wywołaniem jakiejkolwiek funkcji w bloku *while(1)*.

2.0.1.4. Timer

Kolejnym sposobem na komunikację z mikro kontrolerem (lub systemem operacyjnym) jest użycie dostępnego Timera. Polega to na wywołaniu funkcji co określony czas, którego zarządzaniem zajmuje się urządzenie (lub system operacyjny). ROzwiązanie to jest bardzo podobne do odpytywania w pętli, a następnie wywołania funkcji *delete()* z tą różnicą, że użycie timera jest

[5]Wartość w środowisku arduino równoznaczna z napięciem 5V na określonym pinie

dokładniejsze ponieważ wykorzystuje zegar czasu rzeczywistego znajdującego się w CPU[6].

Listing 2.7. Przykładowe użycie timer w środowisku Arduino

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

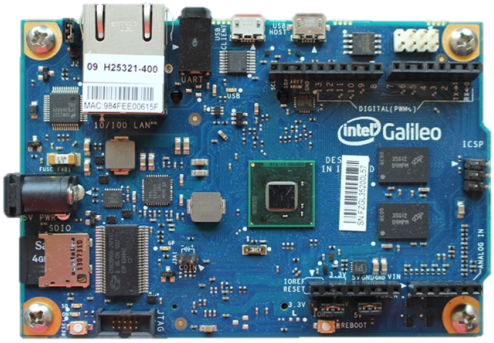
[6]ang. Central Processing Unit - jednostka arytmetyczno-logiczna wykorzystywana do wykonywania obliczeń niezbędnych do działania programu

ROZDZIAŁ 3

Implementacja

3.0.1. Intel Galileo

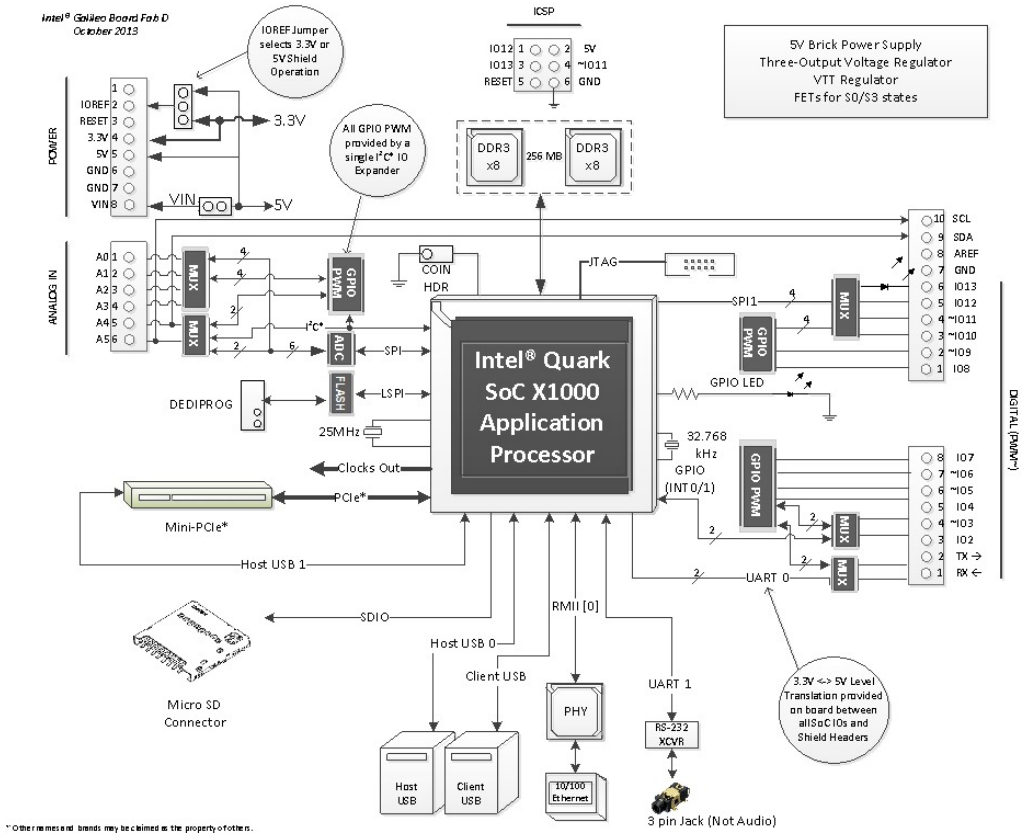
Intel Galileo jest to mikro kontroler oparty na 32-bitowym procesorze Intel® Quark SoC X1000 i taktowaniu 400MHz. Został on wyposażony w 14 pinów cyfrowych (w tym 6 pinów mogących pełnić funkcję PWM[1]) oraz 6 pinów cyfrowych. Każdy z tych pinów jest w stanie operować napięciem od 0V do max 5V. Bardzo dużym atutem Galileo jest wbudowana karta sieciowa, port RS-232, port USB oraz slot karty microSD. Galileo może być używane w dwóch trybach - trybie w pełni kompatybilnym z Arduino oraz w trybie z zainstalowanym systemem operacyjnym (np. Linux).



Rysunek 3.1. Galileo Gen 1 Board

Źródło: http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g1-datasheet.html

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



Rysunek 3.2. Schemat logiczny układu Intel Galileo

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

Do komunikacji z Galileo programista może użyć portu RS-232, USB (działające w trybie host oraz client) oraz wyjścia Ehernet. Intel Galileo jest zasilane napięciem 5V i 2.0A, które może być dostarczone poprzez zasilacz dostarczony do zestawu lub poprzez podłączenie zasilania do portów PWR[2] oraz GND[3]. Najprostszym sposobem na uruchomienie programu jest skompilowanie go za pomocą środowiska Arduino Studio, pisząc w języku C i używając dostarczonych przez producenta Arduino funkcji do obsługi portów, a następnie przesłania skompilowanej wersji poprzez kabel USB do

[2]Port używany jako port zasilania(5V)
[3]Port używany jako masa

urządzenia. Po przesłaniu program zostaje automatycznie uruchomiony. Gdy mamy zainstalowany system operacyjny wtedy komunikację można prowadzić w dowolnym języku programowania.

3.1. Protokół komunikacyjny I^2C

I^2C[4] 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 *SDA*[5] oraz jednokierunkowa linia *SCL*[6]. Każdą transmisję danych należy rozpocząć sygnałem *START* oraz zakończyć sygnałem *STOP*. Dane wysyłane są od najstarszego do najmłodszego bitu oraz otrzymanie każdego z nich musi być potwierdzone przez odbiornik (bit ACK[7]). Należy również pamiętać, aby każdą komunikację z urządzeniem rozpocząć i zakończyć ustawiając linie *SDA* oraz *SCL* w stan nieaktywny (HIGH). Podstawowymi zaletami protokołu są:

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

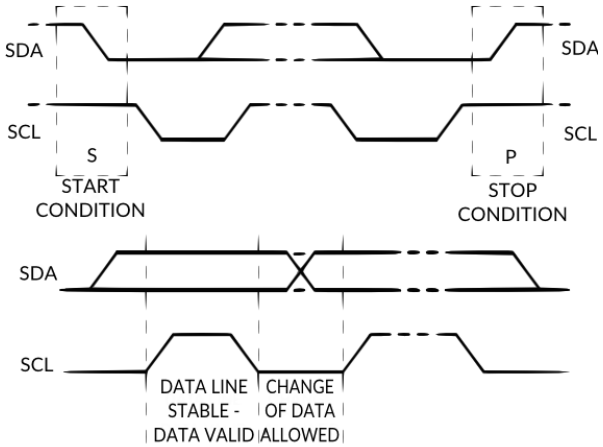
2. Duża dostępność sprzętu w sklepach

3. Transmisja jest odporna na zakłócenia zewnętrzne

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

Nazwa I^2C jest nazwą zastrzeżoną dlatego też w literaturze bardzo często spotyka się określenie *TWI*[8]. Jest ono stosowane w mikro kontrolerach firmy *Atmel*.

[4] ang. Inter-Integrated Circuit
[5] ang. Serial Data Line
[6] ang. Serial Clock Time
[7] ang. Acknowledge
[8] ang. Two Wire Interface



Rysunek 3.3. Przebieg czasowy protokołu I^2C

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

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

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

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

Adresowanie urządzenia odbywa się poprzez wysłanie pojedynczych bitów adresu (pamiętając o kolejności MSB->LCB[9]) oraz wygenerowanie impulsu zegara. Gdy chcemy zaadresować urządzenie, którego adresem jest np. 4 należy wykonać:

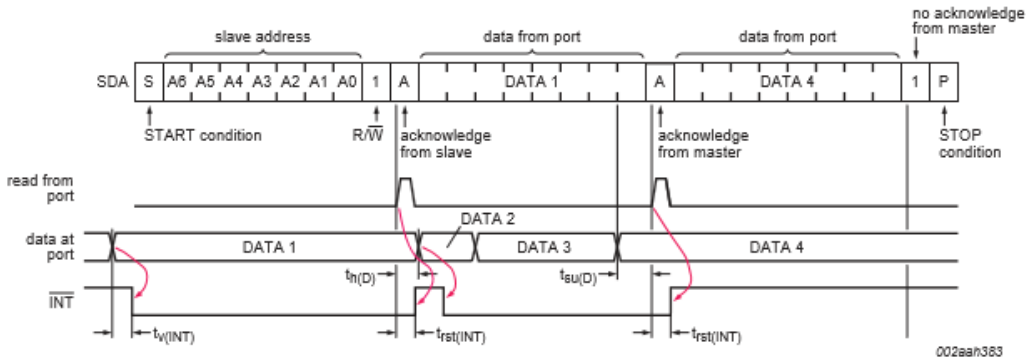
[9]Wysyłanie odbywa się w kolejności od najbardziej znaczących (najstarszych) bitów

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

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

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

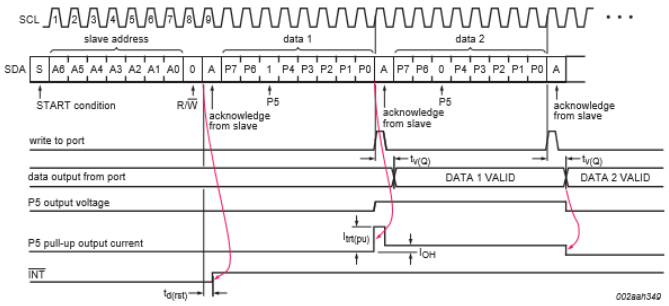
Po otrzymaniu potwierdzenia na linii *SDA* można zacząć czytać dane przesyłane z urządzenia.



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

Źródło: Karta katalogowa I/O Expander PCF8574N

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



Rysunek 3.5. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

3.1.2. Problemy z bibliotekami

3.1.3. Własna implementacja I^2C

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

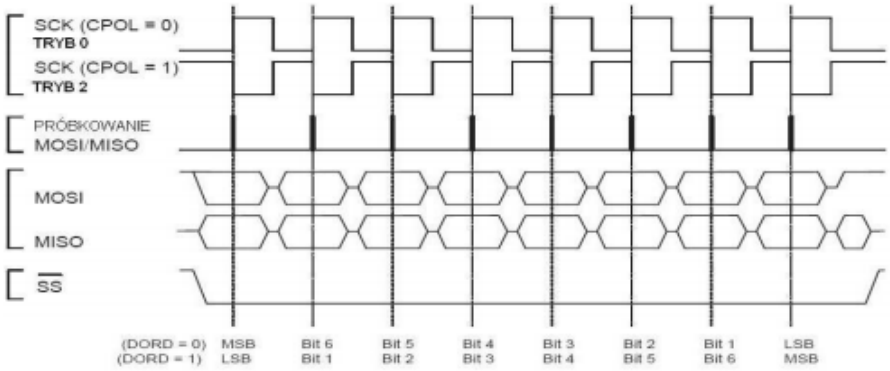
3.2. Protokół komunikacyjny SPI

Kolejnym przykładem interfejsu szeregowego jest protokół SPI[10]. Składa się on z czterech podstawowych linii - dwóch służących do przesyłania danych w przeciwnych kierunkach, jednej z sygnałem taktującym synchronizującym transfer danych oraz linii chip select.

Linia MISO[11] jest linią wejścia danych dla urządzenia nadrzędnego (master), a wyjściem dla urządzenia podrzędnego (slave), linia MOSI[12] jest wyjściem dla urządzenia master, a wejściem dla slave. Linia SCK[13] jest wejściem taktującym zegar. Sygnał taktujący jest zawsze generowany przez układ ma-

[10] ang. Serial Peripherial Interface
[11] ang. Master In Slave Out
[12] ang. Master Out Slave In
[13] ang. Serial Clock

ster. 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.[?]



Rysunek 3.6. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0

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

3.2.1. Komunikacja poprzez protokół SPI

Komunikacja zawsze przebiega dwustronnie (duplex[14]). 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 protokółu I^2C w wydaniu z 2012r. to 5-MHz, gdy prędkość SPI jest praktycznie nieograniczona.

[14]Nadawanie i odbieranie informacji odbywa się w obu kierunkach

3.2.2. VM800B

FDTI EVE VM800B jest to wyświetlacz dotykowy wraz z wbudowanym kontrolerem audio. Podstawowe cechy urządzenia:

1. Pojedynczy układ scalony dla wyświetlacza oraz kontrolera Audio
2. Ekran 3.5" LCD
3. 262 tys. kolorów
4. Możliwość wygładzania krawędzi
5. Możliwość komunikacji poprzez użycie interfejsu I^2C lub SPI
6. Wbudowane widgety dostępne dla użytkownika
7. Zakres pracy wyświetlacza: $-40°C$ do $85°C$

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.

3.3. Założenia funkcjonalne projektu

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

3.3.1. Symulator samochodu

Do celów implementacjach komputer nie mógł zostać zamontowany do fizycznego samochodu. Do celów testowych zstał zabudowany symulator samochodu składający się z:

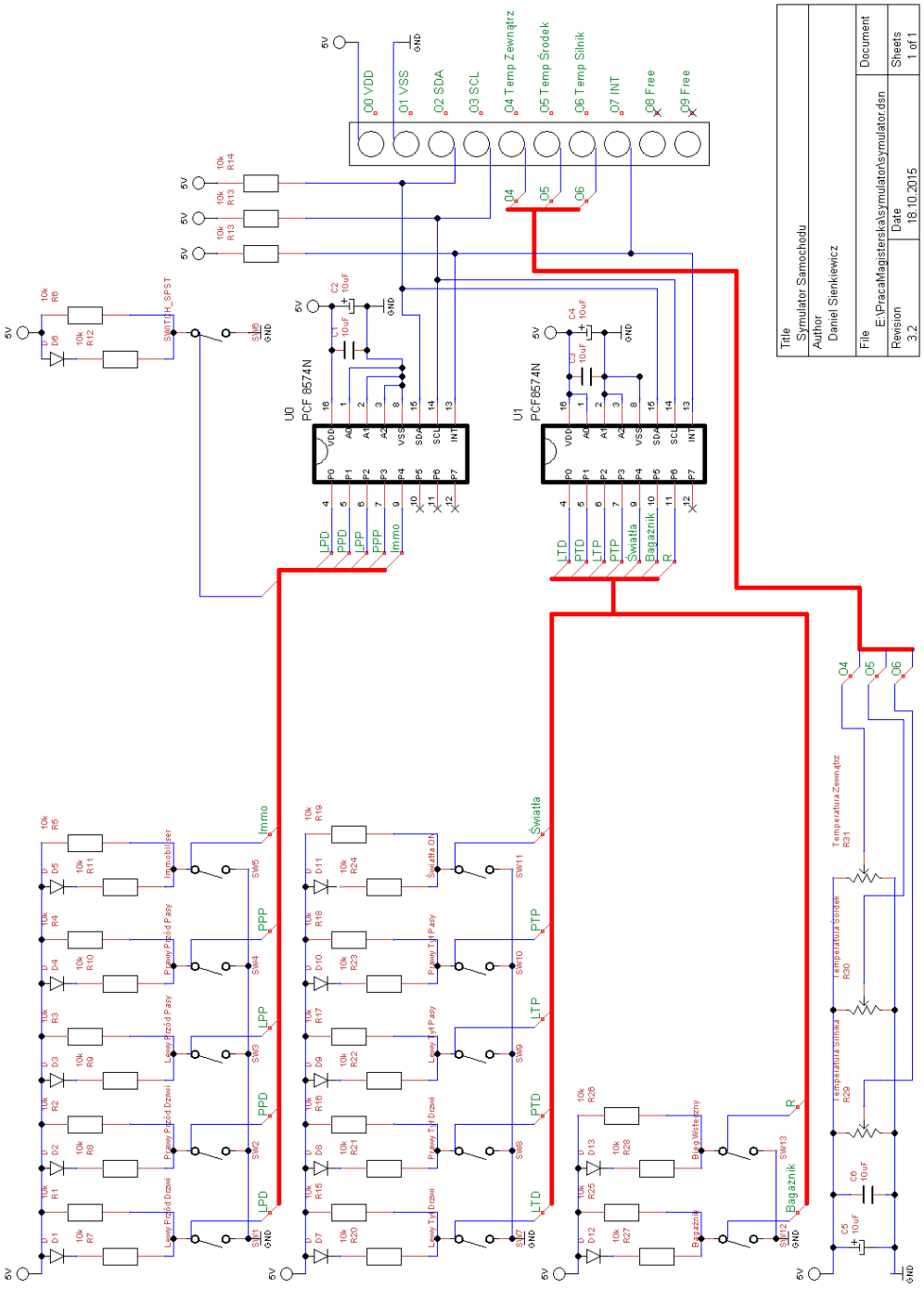
1. 5 guzików symulujących drzwi samochodu (wraz z bagażnikiem)
2. 4 guzików symulujący pasy pasażerów

3. 3 potencjometrów symulujących czujniki temperatury w samochodzie
4. 1 guzika symulującego wrzucenie biegu wstecznego
5. 1 guzika symulującego włączenie świateł w samochodzie

Guziki wysyłają sygnały cyfrowe, a potencjometry sygnały analogowe. Komunikacja z Galileo odbywa się poprzez użycie protokołu I^2C wykorzystywane poprzez *I/O Expander PCF 8574N* oraz 3 linii wyprowadzonych dla sygnałów analogowych.

3.4. Dalsze kroki oraz propozycje

schemat blokowy z BAJERAMI i wybrane to co zrobię



Rysunek 3.7. Schemat elektryczny symulatora samochodu

Źródło: Opracowanie własne

Zakończenie

TO DO

DODATEK A

Karty Katalogowe

Katalog *datasheets* zawiera karty katalogowe użytych podzespołów

DODATEK B

Porównanie dostępnych na rynku mikro kontrolerów

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

Tabela B.1. Specyfikacja dostępnych na rynku mikro kontrolerów

Źródło: http://eu.mouser.com/applications/open-source-hardware-galileo-pi/

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

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

DODATEK D

Programy

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

Spis tabel

B.1. Specyfikacja dostępnych na rynku mikro kontrolerów 25

C.1. Mapowanie portów Intel Galileo na pliki w systemie Linux . . 26

Spis rysunków

3.1. Galileo Gen 1 Board 13
3.2. Schemat logiczny układu Intel Galileo 14
3.3. Przebieg czasowy protokołu I^2C 16
3.4. Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N 17
3.5. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N 18
3.6. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0 19
3.7. Schemat elektryczny symulatora samochodu 22

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