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,

Dostępne dla użytkownika opcje systemu to:

1. Informacja na temat aktualnej temperatury w samochodzie, na zewnątrz oraz w silniku,
2. Informacja na temat stanu drzwi (otwarte/zamknięte),
3. Informacja na temat stanu pasów (zapięte/odpięte),
4. Aktualna pozycja GPS wraz z możliwością zapisu tej pozycji na kracie pamięci,
5. Ekran z obrazem służący jako inteligentne lusterko wsteczne,

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. Intel Galileo 7
2.0.2. FDTI EVE VM800B 9
2.0.3. Symulator samochodu 10

3. Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem 12
3.0.1. Odpytywanie w pętli 12
3.0.2. Timer 12
3.0.3. Porty 13
3.0.4. Przerwania 15

4. Implementacja 18
4.1. Protokół komunikacyjny I^2C 18
4.1.1. Użycie protokołu I^2C na przykładzie I/O Expander PCF8574N 19
4.2. Protokół komunikacyjny SPI 21
4.2.1. Komunikacja poprzez protokół SPI 22

5. Działanie komputera pokładowego 24
5.1. Założenia funkcjonalne projektu 24
5.2. Opis działania 24
5.3. Wnioski oraz własne doświadczenia 28

Zakończenie 30

A. Karty Katalogowe 31

B. Porównanie dostępnych na rynku mikro kontrolerów 32

C. Mapowanie portów Intel Galileo na pliki w systemie Linux 33

D. Programy 34

Bibliografia 35

Spis tabel 36

Spis rysunków 37

Oświadczenie 38

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, w środku oraz aktualnym stanie pasów i drzwi. Ponadto powinien on móc zapisać aktualną pozycję *GPS* na karcie pamięci microSD. Komunikacja użytkownika z komputerem będzie odbywała się poprzez użycie ekranu dotykowego *FTDI EVE VM800B*.

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 zainstalownym systemem operacyjnym Linux YOCTO,
2. Ekran dotykowy FTDI EVE VM800B jako interfejs komunikacyjny komputera z użytkownikiem,
3. Kod został skompilowany przy użyciu Arduino studio oraz natywnego systemu Linux YOCTO,
4. 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

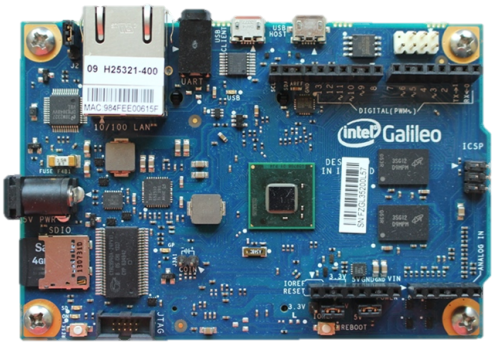
TO DO

ROZDZIAŁ 2

Architektura

2.0.1. Intel Galileo

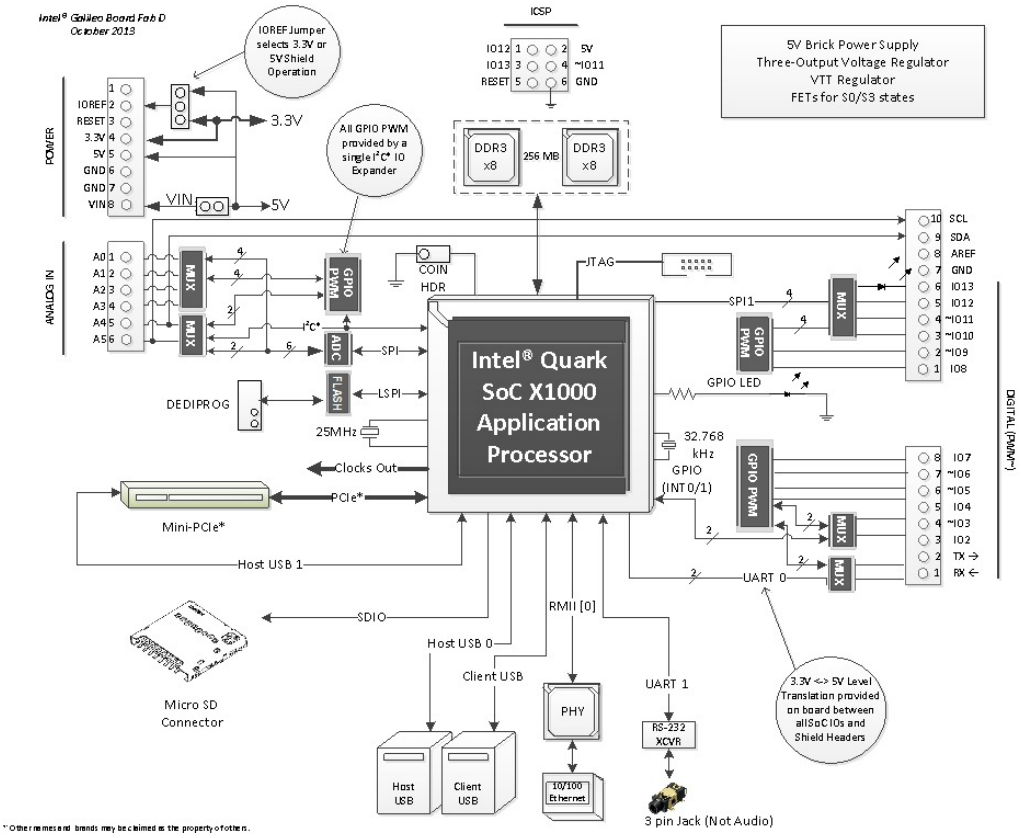
Intel Galileo jest to mikro kontroler oparty na 32-bitowym procesorze Intel® Quark SoC X1000 i taktowaniu 400MHz. Został on wyposażony w 14 pinów cyfrowych (w tym 6 pinów mogących pełnić funkcję PWM[1]) oraz 6 pinów cyfrowych. Każdy z tych pinów jest w stanie operować napięciem od 0V do max 5V. Bardzo 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).[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]

[1]ang. 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]

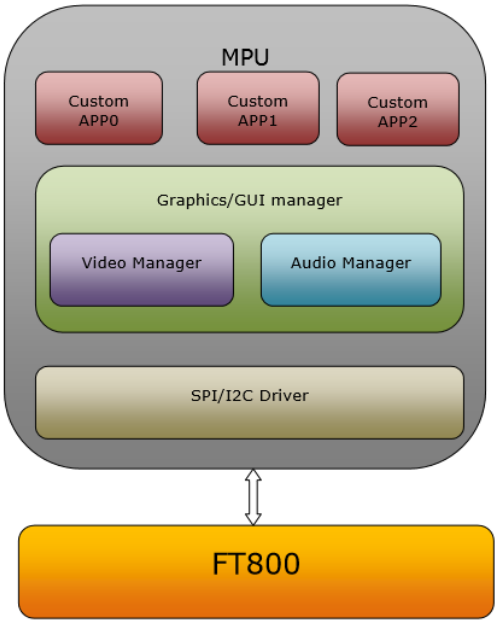
Do komunikacji z Galileo programista może użyć portu RS-232, USB (działające w trybie host oraz client) oraz wyjścia Ehernet. Intel Galileo jest zasilane napięciem 5V i 2.0A, które może być dostarczone poprzez zasilacz z zestawu lub poprzez podłączenie zasilania do portów PWR[2] oraz GND[3]. Najprostszym 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 urządzenia. Po prze-

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

słaniu program zostaje automatycznie uruchomiony. Gdy mamy zainstalowany system operacyjny wtedy komunikację można prowadzić w dowolnym języku programowania łącząc się z systemem poprzez SSH[4] lub port RS-232.

2.0.2. FDTI EVE VM800B

FDTI EVE VM800B jest to wyświetlacz dotykowy wraz z wbudowanym kontrolerem audio.



Rysunek 2.3. Architektura Ekranu FTDI EVE VM800B

Źródło: FT800 Programmers Guide

Podstawowe cechy urządzenia[6]:

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

[4]ang. 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. Wbudowane widgety dostępne dla użytkownika
7. Zakres pracy wyświetlacza: $-40°C$ do $85°C$

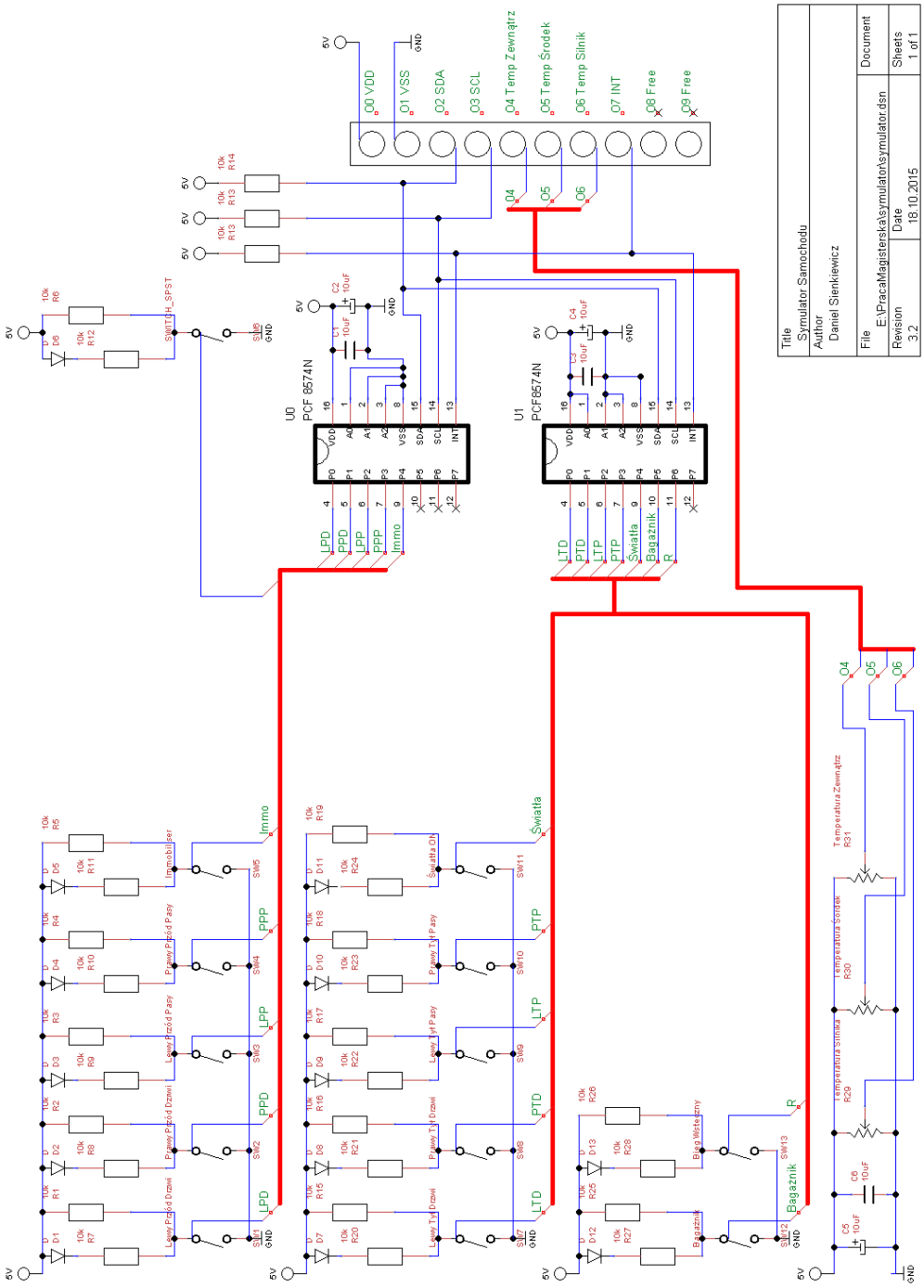
Komunikacja Galileo z Ekranem odbywa się poprzez protokół komunikacyjny *SPI*. Do tego celu została napisana własna wersja funkcji służących do wysłania oraz odczytania danych z ekranu.

2.0.3. Symulator samochodu

Do celów implementacjach komputer nie mógł zostać zamontowany do fizycznego samochodu. Z tego powodu został 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.



Rysunek 2.4. Schemat elektryczny symulatora samochodu

Źródło: Opracowanie własne

ROZDZIAŁ 3

Mechanizmy komunikacji systemu mikroprocesorowego z otoczeniem

3.0.1. 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 3.1. 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)*.

3.0.2. 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 *delay()* z tą różnicą, że użycie timera jest

dokładniejsze ponieważ wykorzystuje zegar czasu rzeczywistego znajdującego się w CPU[1].

Listing 3.2. Przykładowe użycie timer w środowisku Arduino

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

3.0.3. 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 ustalić tryb w jakim ma on działać (wejście/wyjście) a następnie wysłać/odczytać dane. Dla portu analogowego wygląda to następująco:

Listing 3.3. Obsługa portu analogowego w środowisku Arduino

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

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

oraz odpowiednio dla portu cyfrowego:

Listing 3.4. 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 3.5. Obsługa portu cyfrowego w środowisku Linux (język C)

```
FILE *fp;
int value;

// Ustawienie portu cyfrowego nr 13 jako port wyjscia
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 3.6. 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 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
```

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

3.0.4. 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[2])
   (b) Maskowalne
3. Wyjątek

Przerwania programowe wywołuje się za pomocą komendy *INT XX*, gdzie *XX* oznacza numer przerwania zadeklarowanego w tablicy wektorów przerwań[3], 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*[4].

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

Listing 3.7. 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

[2]Non-Maskable Interrupt

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

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

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[5]
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[6]
4. FALLING - przerwanie zostanie zgłoszone gdy wartość na określonym pinie zostanie zmieniona z HIGH na LOW

[5]Wartość w środowisku arduino równoznaczna z napięciem 0V na określonym pinie
[6]Wartość w środowisku arduino równoznaczna z napięciem 5V na określonym pinie

ROZDZIAŁ 4

Implementacja

4.1. Protokół komunikacyjny I^2C

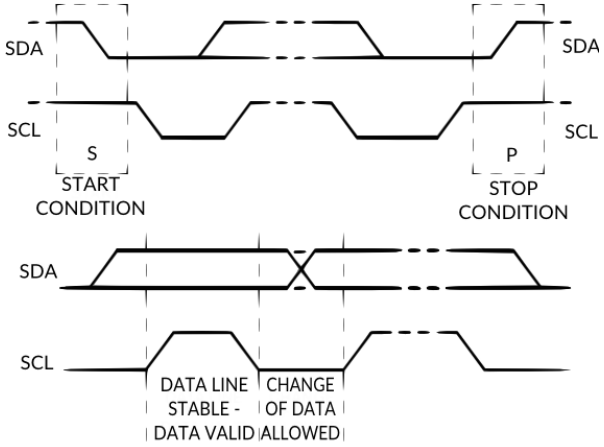
I^2C[1] jest szeregowym interfejsem stworzonym przez firmę *Philips* służącym do przesyłania danych między urządzeniami elektrycznymi.

Podstawową cechą I^2C jest wykorzystywanie dwóch linii służących do komunikacji: dwukierunkowa linia *SDA*[2] oraz jednokierunkowa linia *SCL*[3]. 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[4]). 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*[5]. Jest ono stosowane w mikro kontrolerach firmy *Atmel*.

[1] ang. Inter-Integrated Circuit
[2] ang. Serial Data Line
[3] ang. Serial Clock Time
[4] ang. Acknowledge
[5] ang. Two Wire Interface

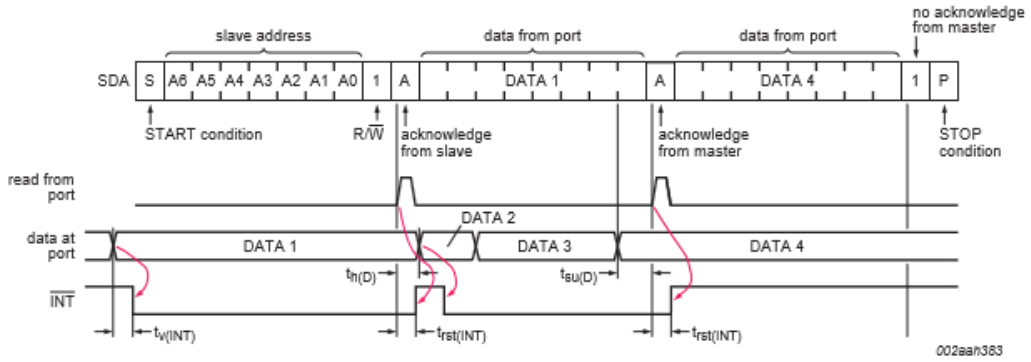


Rysunek 4.1. Przebieg czasowy protokołu I^2C

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

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

W symulatorze ze względu na dużą ilość wychodzących sygnałów zastosowano komunikacje poprzez I^2C. Do obsługi tego został zamontowane dwa I/O Expandery 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 I^2C co znacznie ułatwiło dalsze korzystanie z Galileo ze względu na pozostałe wolne porty cyfrowe, które będą potrzebne w dalszej części do komunikacji z ekranem. Zaletą tego rozwiązania jest możliwość późniejszego podłączenia większej ilości czujników bez jakiejkolwiek ingerencji w okablowanie Galileo. Odbieranie danych rozpoczyna się wysłaniem sygnału *START*, a następnie zaadresowaniu urządzenia. Kolejnym krokiem jest ustalenie trybu w jakim mają działać I/O Expandery (w tym wypadku *READ*) oraz odczytanie potwierdzenia. Po wykonaniu tych czynności można rozpocząć odbieranie danych, które należy zakończyć wysłaniem sygnału *STOP*.



Rysunek 4.2. Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N

Źródło: Karta katalogowa I/O Expander PCF8574N

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

Adresowanie urządzenia odbywa się poprzez wysłanie pojedynczych bitów adresu (pamiętając o kolejności MSB->LCB[6]) oraz wygenerowanie impulsu zegara. Gdy chcemy zaadresować urządzenie, którego adresem jest np. 4 należy wykonać:

Listing 4.1. Adresowanie urządzenia I^2C na przykładzie PCF8574N

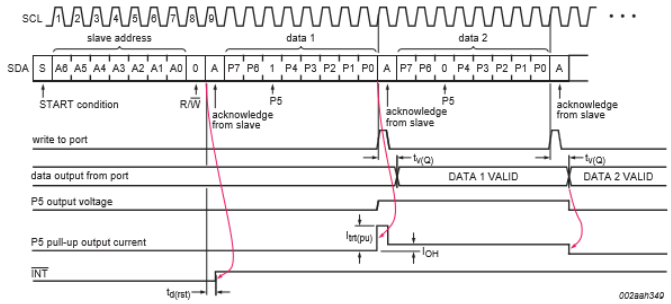
```
int adres = 4;
for(m = 0x80; m; m >>= 1){
    if(adres & m)
      digitalWrite(sda, HIGH);
    else
      digitalWrite(sda, LOW);
```

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

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

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

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



Rysunek 4.3. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N

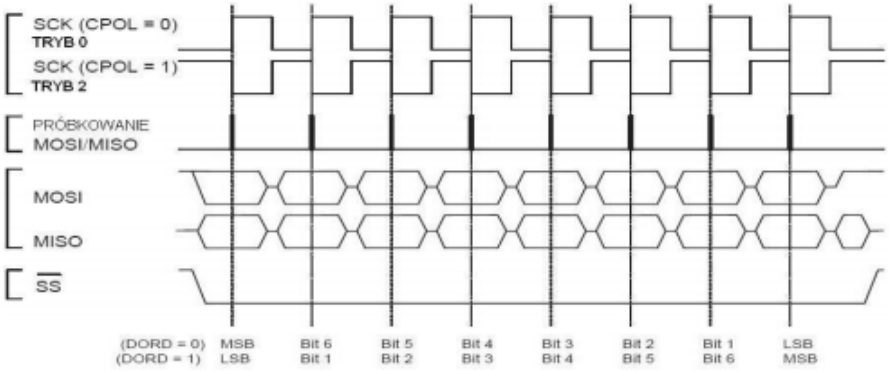
Źródło: Karta katalogowa I/O Expander PCF8574N

4.2. Protokół komunikacyjny SPI

Kolejnym przykładem interfejsu szeregowego jest protokół SPI[7]. 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*.

[7] ang. Serial Peripherial Interface

Linia MISO[8] jest linią wejścia danych dla urządzenia nadrzędnego (master), a wyjściem dla urządzenia podrzędnego (slave), linia MOSI[9] jest wyjściem dla urządzenia master, a wejściem dla slave. Linia SCK[10] 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 4.4. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0

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

4.2.1. Komunikacja poprzez protokół SPI

Komunikacja zawsze przebiega dwustronnie (duplex[11]). 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

[8] ang. Master In Slave Out
[9] ang. Master Out Slave In
[10] ang. Serial Clock
[11] Nadawanie i odbieranie informacji odbywa się w obu kierunkach

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.

ROZDZIAŁ 5

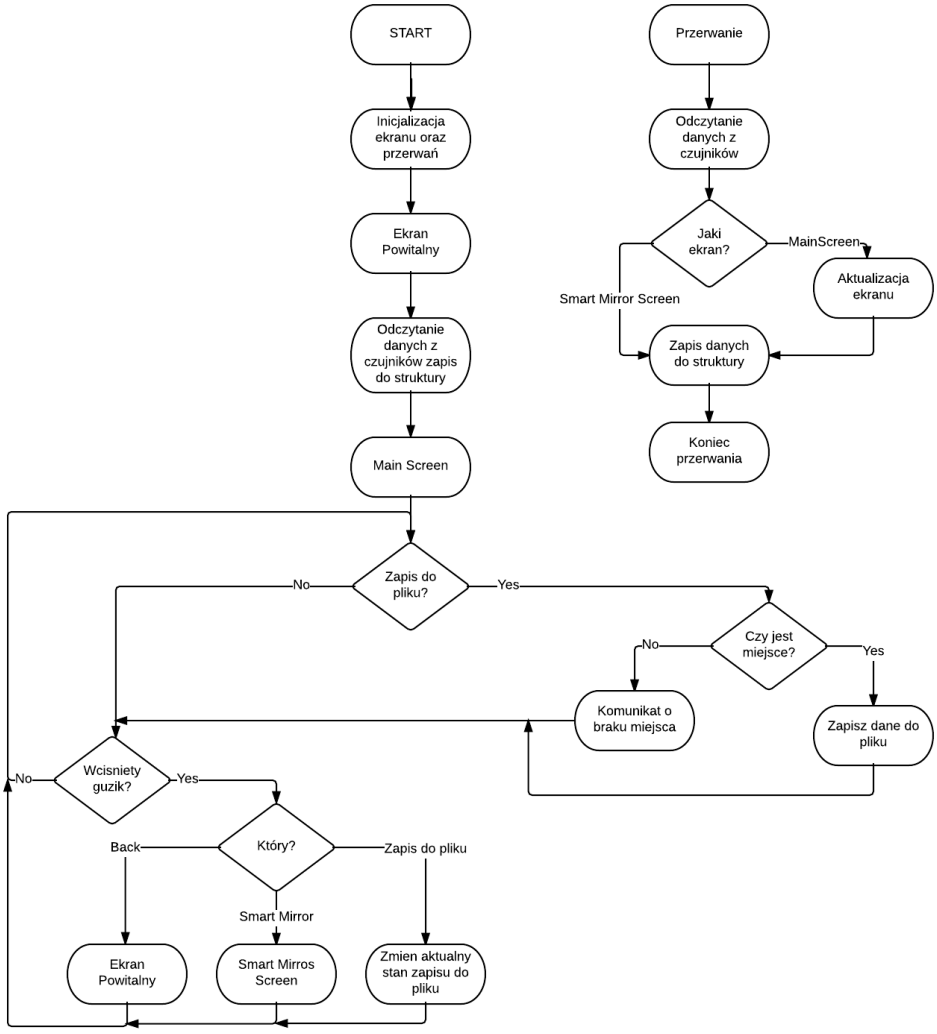
Działanie komputera pokładowego

5.1. 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, włączenia-/wyłączenia świateł oraz czujniki temperatury. Dodatkowym elementem była komunikacja z zewnętrznym ekranem dotykowym służącym do komunikacji pomiędzy użytkownikiem a komputerem.

5.2. 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. Użytkownik po wejściu do samochodu oraz włączeniu zapłonu powoduje automatyczny start systemu. Na ekranie pojawia się powitanie oraz ekran startowy, na którym można zobaczyć aktualną pozycję 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.



Rysunek 5.1. Schemat blokowy programu

Źródło: Opracowanie własne

Komunikacja z Intel Galileo z ekranem odbywa się używając protokołu SPI. Łącznie użytych zostało 7 linii: SDI, SDO, clock, PD, CS[1] oraz PWR[2] i GND[3]. Rozpoczęcie pracy wyświetlacza polega na wpisaniu określonych wartości do pewnych obszarów jego pamięci określając w ten sposób np. rozdzielczość lub włączenie/wyłącznie modułu odpowiedzialnego za dźwięk czy dotyk.

Następnie należy wywołać zestaw funkcji odpowiedzialny 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:

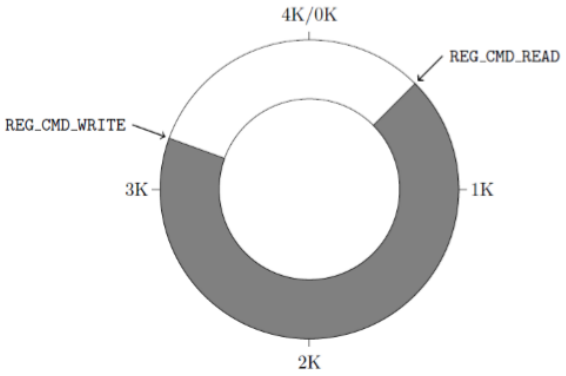
1. Poczekaj aż wszystko co miało zostać wyświetlone, zostanie wyświetlone - opróżnij bufor
2. Określ co będziemy rysować - np. linia lub kropka i dodaj to do bufora
3. Przesuń się o 4 bity w buforze
4. Ustaw wszystkie potrzebne parametry - np. wielkość, kolor lub położenie i dodaj to do bufora
5. Przesuń się o 4 bity w buforze
6. Wyświetl wszystko do zostało dodane do bufora

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

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

Listing 5.1. Narysowanie linii na ekranie

[1]Linia Chip select wyświetlacza
[2]ang. Power - 5V
[3]ang. Ground



Rysunek 5.2. Bufor dostępny podczas programowania ekranu VM800

Źródło: FT800 Programmers Guide

```
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 5.2. 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);
}
```

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.

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 tekstowego do momentu wyłączenia systemu, wyczerpania miejsca na karcie lub zmiany tej opcji przez użytkownika.

5.3. 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. 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
2. Sterowanie głośnością radia w zależności od prędkości samochodu
3. Blokada immobilizer
4. Obsługa telefonu komórkowego za pomocą bluetooth
5. 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* zawier kod źródłowy oprogramowania stworzonego na potrzeby pracy.

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 32

C.1. Mapowanie portów Intel Galileo na pliki w systemie Linux . . 33

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. Schemat elektryczny symulatora samochodu 11

4.1. Przebieg czasowy protokołu I^2C 19
4.2. Przykładowy schemat odbierania danych poprzez I^2C na przykładzie PCF8574N 20
4.3. Przykładowy schemat wysyłania danych poprzez I^2C na przykładzie PCF8574N 21
4.4. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0 22

5.1. Schemat blokowy programu 25
5.2. Bufor dostępny podczas programowania ekranu VM800 27

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