Politechnika Świętokrzyska w Kielcach

Wydział Elektroniki, Automatyki i Informatyki

Projekt: **Programowanie w języku C2**

Grupa:	Temat:	Skład grupy:
2ID14B	**PC Master Control Center – Centrum sterowania oświetleniem LED i wyświetlaczem OLED**	**Michał Młodawski**
Rok studiów:		**Konrad Nowakowski**
2		

PC Master Control Center – Centrum sterowania oświetleniem LED i wyświetlaczem OLED

Opracowanie i sprawozdanie projektu PCMCC

Grudzień 2017

Wersja dokumentu 16122017-005PL

Spis treści

1. Opis projektu

Projekt PCMC (PC Master Control Center) powstał w celu wykorzystania możliwość nowych technologii połączonych razem ze sprzętami peryferyjnymi. Dodatkowym założeniem projektu było napisanie aplikacji pulpitowej, która będzie komunikować się przy pomocy standardu UART (universal asynchronous receiver-transmitter) z modułem LED oraz modułem OLED i będzie dostępna w wielu wersjach językowych wspierana przez platformę dystrybucji cyfrowej Steam.

2. Wykorzystane technologie

W naszym projekcie wykorzystaliśmy najnowocześniejsze technologie dla osiągnięcia jak najlepszych walorów estetycznych aplikacji pulpitowej i szybkości działania modułów LED oraz OLED.

1. Użyte języki programowania oraz formaty tekstu

1. Asembler do sprzętowego opóźnienia działania funkcji wyświetlania danych na ekranie OLED.
2. C++ z nakładką QT był wykorzystywany przy stworzeniu Back-End i Front-End aplikacji oraz przy zaprogramowaniu modułu LED/OLED.
3. JavaScript z nakładką QML do zaprogramowania interfejsu graficznego aplikacji.
4. JSON jako format tekstowy dla przejrzystego uporządkowania danych w plikach językowych.

2. Wykorzystane oprogramowanie przy projektowaniu i wdrażaniu projektu

1. AVRDUDE służył jako sterownik do programowania mikrokontrolerów .
2. Device Monitoring Studio do analizowania ruchu między modułami a oprogramowaniem.
3. Doxygen do prowadzenia dokumentacji kodu.
4. MkAvrCalculator aplikacja wspierała proces wyliczania Fusebitów dla mikrokontrolerów.
5. Qt Creator jako główne IDE do programowania aplikacji.
6. Visual Studio 2017 jako kompilator i debuger dla QT.
7. Autodesk EAGLE program wykorzystany do wspomagania projektowania obwodów elektrycznych.

3. Wykorzystane układy scalone i inne elementy aktywne w projekcie

1. Attiny85-20PU jako sterownik modułu LED.
2. ATmega328P-PN jako sterownik modułu OLED.
3. Pasek 8 diod WS2812 do wyświetlania kolorów.
4. MCP2221 mostek USB służący do emulacji standardu komunikacji UART.
5. Wyświetlacz OLED REG010016AWPP5N0 służący do wyświetlania komunikatów.
6. DS1307N+ Zegar czasu rzeczywistego.
7. 24LC512-I/P Pamięć EEPROM do zapisu flag, grafiki, czy kolorów.

3. Użyte biblioteki w projekcie

1. Biblioteki zewnętrzne

LP.	Nazwa biblioteki	Licencja	Opis
1	Arduino.h	GNU Lesser General Public	Framework dodający obsługę pinów cyfrowych.
2.	DS1307.h	GNU General Public License version 3	Biblioteka dodaje obsługę zegara DS1307 firmy Maximintegrated™.
3.	SoftwareSerial.h	GNU Lesser General Public	Biblioteka dodaje programową transmisję UART.
4.	Stream.h	GNU Lesser General Public	Biblioteka dodaje możliwość zamiany typów danych.
5.	TwoWire.cpp	GNU Lesser General Public	Biblioteka dodaje możliwość komunikacji z układami według standardu I²C.
6.	Print.h	GNU Lesser General Public	Biblioteka dodaje możliwość wysyłania danych o dowolnych typach danych do różnych portów wyjść.
7.	Oled.h	MIT	Przerobiona biblioteka LiquidCrystal dojąca obsługę OLED. Dodaliśmy kolejną jej modyfikację dodając tryb graficzny. Metody **displayBuf** oraz **readAddress.**
8.	Adafruit_NeoPixel.h	GNU Lesser General Public	Biblioteka dodaje możliwość sterowania diodami LED.
9.	TinyWireM.h	GNU General Public License version 2.1	Biblioteka dodaje możliwość komunikacji z układami według standardu I²C dla mikrokontrolerów z rodziny Attiny.
10.	Steam SDK	Licencja wewnętrzna: http://store.steampowered.com/subscriber_agreement/	Zestaw SDK umożliwiający integrację aplikacji z sklepem dystrybucji cyfrowej.

Tabela 1 Biblioteki zewnętrzne wykorzystane w projekcie.

2. Biblioteki wewnętrzne

1. QByteArray udostępnia tablicę bajtów.
2. QFile udostępnia interfejs do odczytu i zapisu do plików.
3. QFileInfo dostarcza niezależne od systemu informacje o plikach.
4. QGuiApplication zarządza sterowaniem aplikacją GUI i ustawieniami graficznymi.
5. QJsonDocument umożliwia odczyt i zapis dokumentów JSON.
6. QJsonObject hermetyzuje obiekty JSON.
7. QObject jest klasą podstawową wszystkich obiektów Qt.
8. QQmlApplicationEngine zapewnia wygodny sposób ładowania aplikacji z pojedynczego pliku QML.
9. QQmlContext definiuje kontekst w silniku QML.
10. QSerialPort udostępnia funkcje umożliwiające dostęp do portów szeregowych.
11. QSerialPortInfo udostępnia informacje o istniejących portach szeregowych.
12. QString jest klasą, która dostarcza ciąg znaków Unicode.
13. QTextStream zapewnia wygodny interfejs do czytania i pisania tekstu.

4. Funkcjonalność projektu

Część aplikacji projektu PCMC umożliwia za pomocą standardu komunikacyjnego UART, który jest emulowany na standard USB komunikację z modułem LED oraz OLED, które odpowiednio służą do wyświetlania barw na diodach **WS2812** oraz wyświetlania tekstu, grafiki lub zegara na wyświetlaczu OLED. Sterowanie odbywa się za pomocą mikrorozkazów **Tabela 2.**

5. Obsługa projektu i sposób jego uruchomienia

1. Korzystanie z projektu

Aby móc korzystać z projektu należy posiadać konto Steam z dodaną aplikacją i DLC (downloadable content) do swojego konta oraz minimum jeden moduł LED albo OLED. Aplikacja automatycznie wykryje podłączony moduł.

Uwaga! Należy najpierw podłączyć moduły, a następnie uruchomić aplikację.

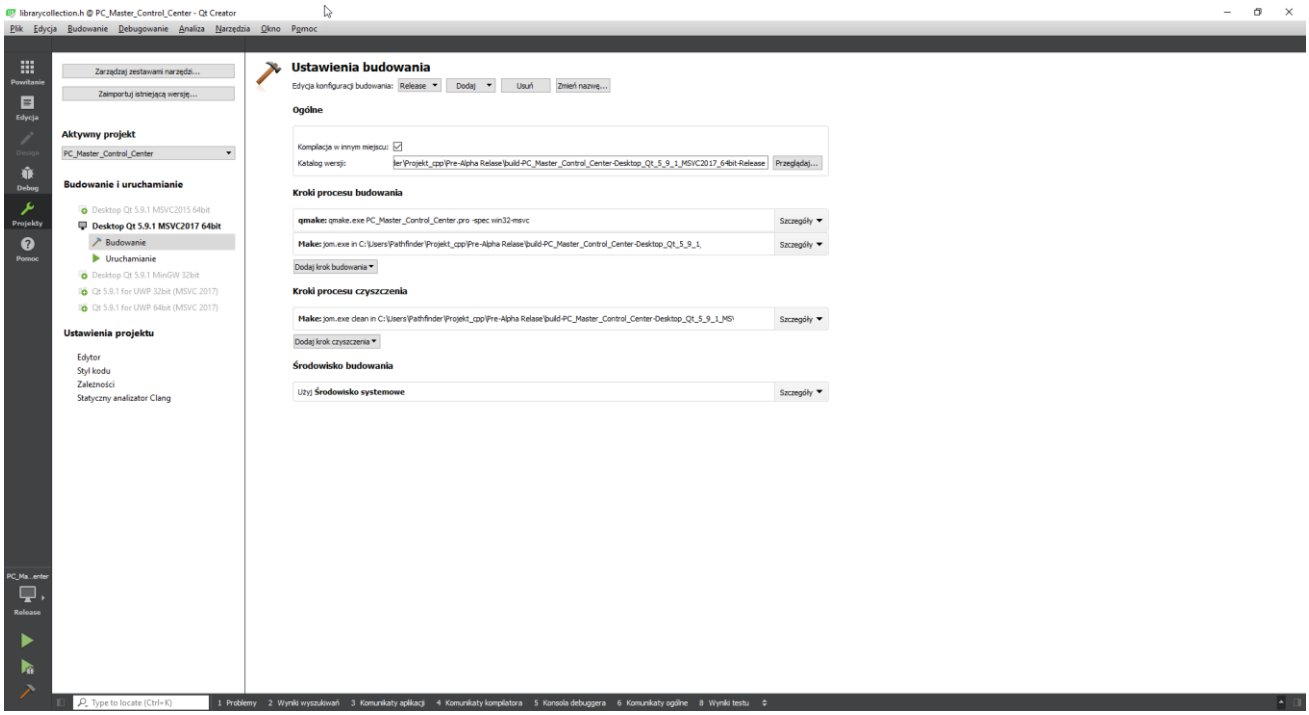
Jeżeli wszystko zakończy się sukcesem powinniśmy mieć dostęp do pełnej wersji aplikacji.

2. Budowanie projektu

Aby móc wybudować projekt musimy posiadać cały pakiet Qt Creator, Windows SDK wersja 10, Debuger oraz kompilator Visual Studio 2017 (msvc2017_64) oraz Avrdude.

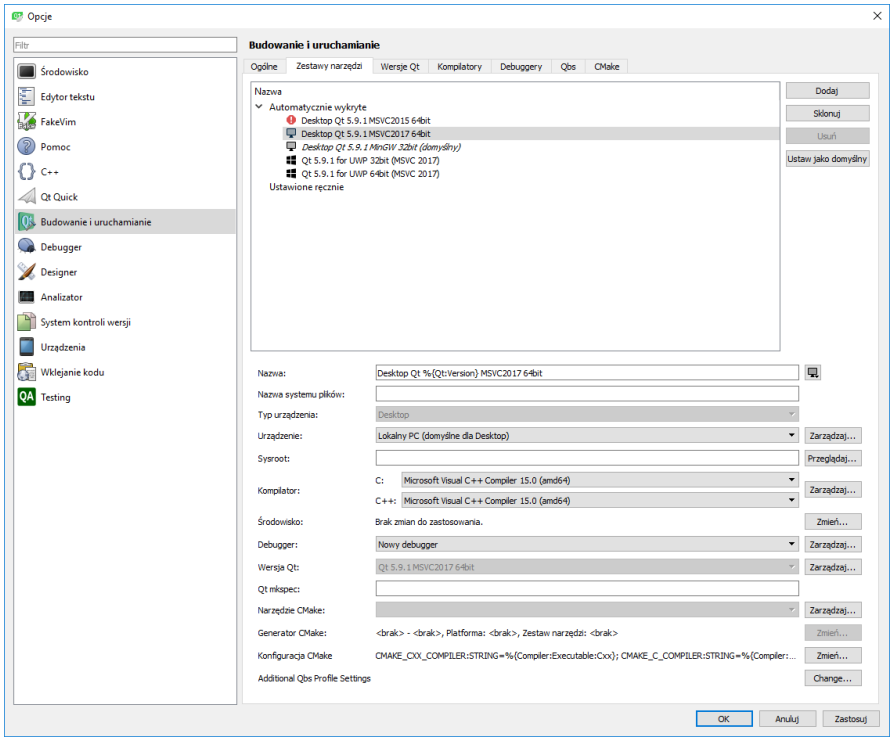
1. Proces kompilacji aplikacji

Pierwszym krokiem jest ustawienie poprawnie Debugera oraz kompilatora:



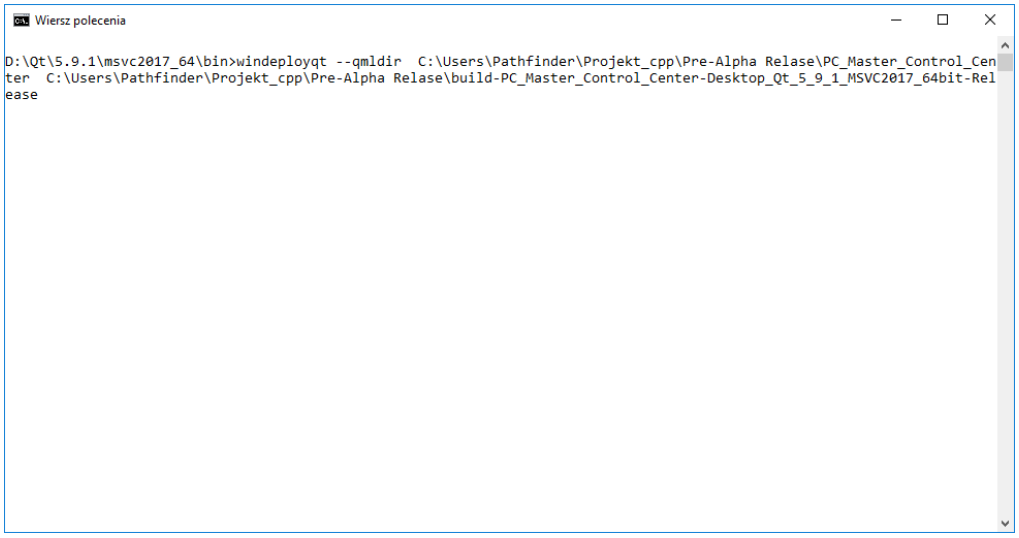
Rysunek 1 Ustawienie kompilatora dla aplikacji.

Jeżeli QT nie wykryje **msvc2017** należy dodać go ręcznie:



Rysunek 2Dodanie debugera jeżeli QT tego za nas nie zrobi.

Po skompilowaniu projektu należy użyć narzędzia **windeployqt.** Służy do zautomatyzowania procesu tworzenia zależności (biblioteki, import QML, wtyczki i tłumaczenia) wymagane do uruchomienia. Tworzy piaskownicę dla środowiska wykonawczego systemu Windows lub drzewo instalacji dla aplikacji pulpitu systemu Windows, które można łatwo dołączyć do pakietu instalacyjnego.

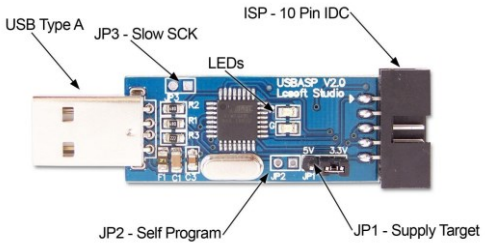


Rysunek 3 Dodanie wymaganych bibliotek i kompilacji QML

Po tych trzech krokach powinniśmy uzyskać w pełni skompilowaną i gotową do uruchomienia aplikację.

2. Proces kompilacji kodu do mikrokontrolera

Podstawowym krokiem jest posiadanie programatora, wyniku wielu testów najlepszy okazał się usbasp 2.0



Rysunek 4 Wykorzystany programator, źródło: https://tosiek.pl

Uwaga! Bez programatora nie jesteśmy w stanie wgrać programu do mikrokontrolera.

Następnie podłączamy według instrukcji przypadku kontrolerów 8-bit wykorzystujemy do tego piny opisane jako **SCK**,**MOSI**,**MISO**,**RESET** i podpinamy zasilanie(**VCC**) oraz masę(**GND**) wyprowadzoną z programatora.

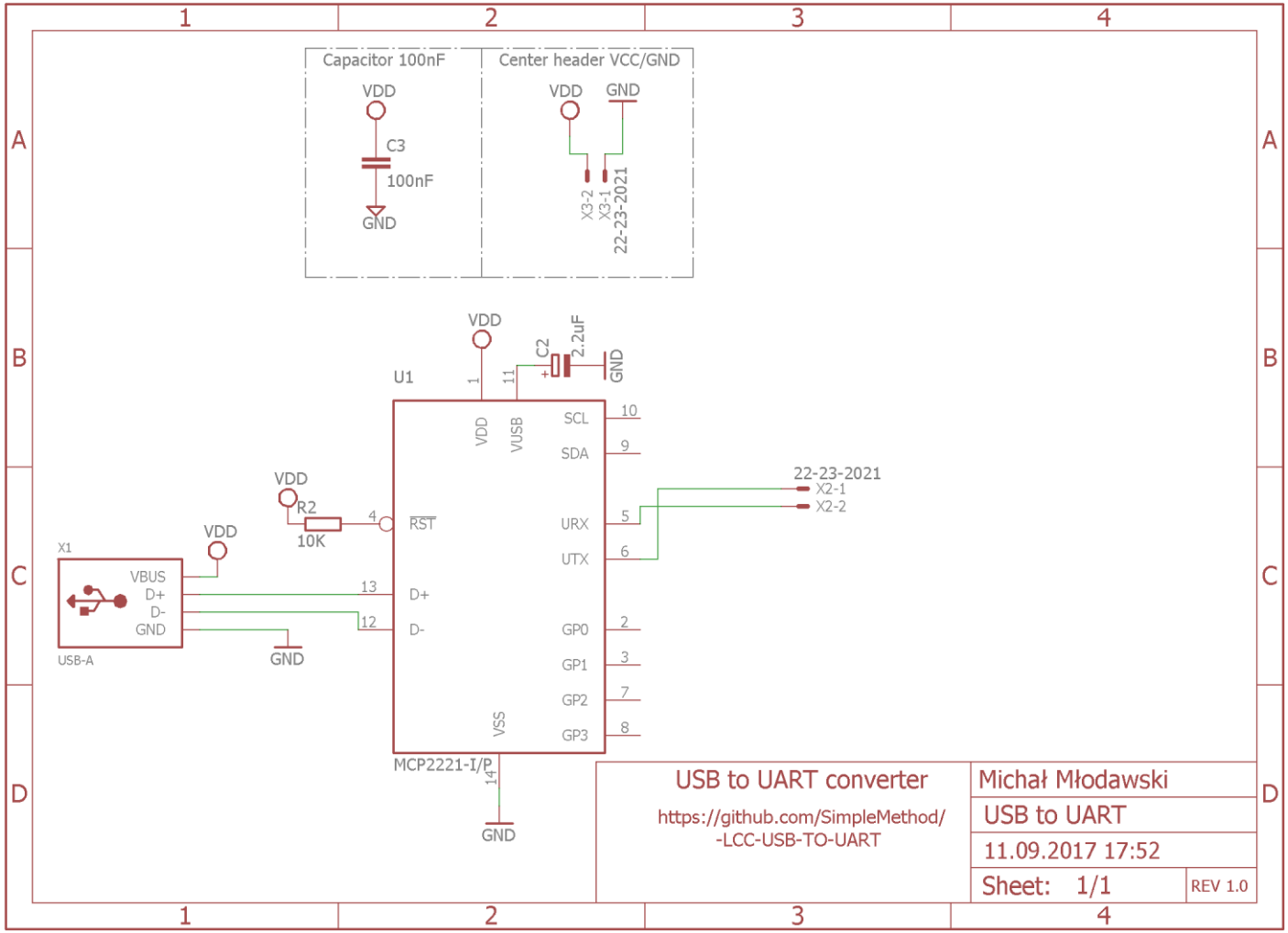
Aplikacja avrdude automatycznie wykryje sterownik. Następnie wprowadzamy argumenty wałowania programu w konsoli CMD lub innym terminalu:

```
avrdude -c usbasp -p m328p -U flash:r:program.hex
```

Jeżeli układ został poprawnie podpięty, to avrdude skompiluje i wgra nasz program do mikroprocesora.

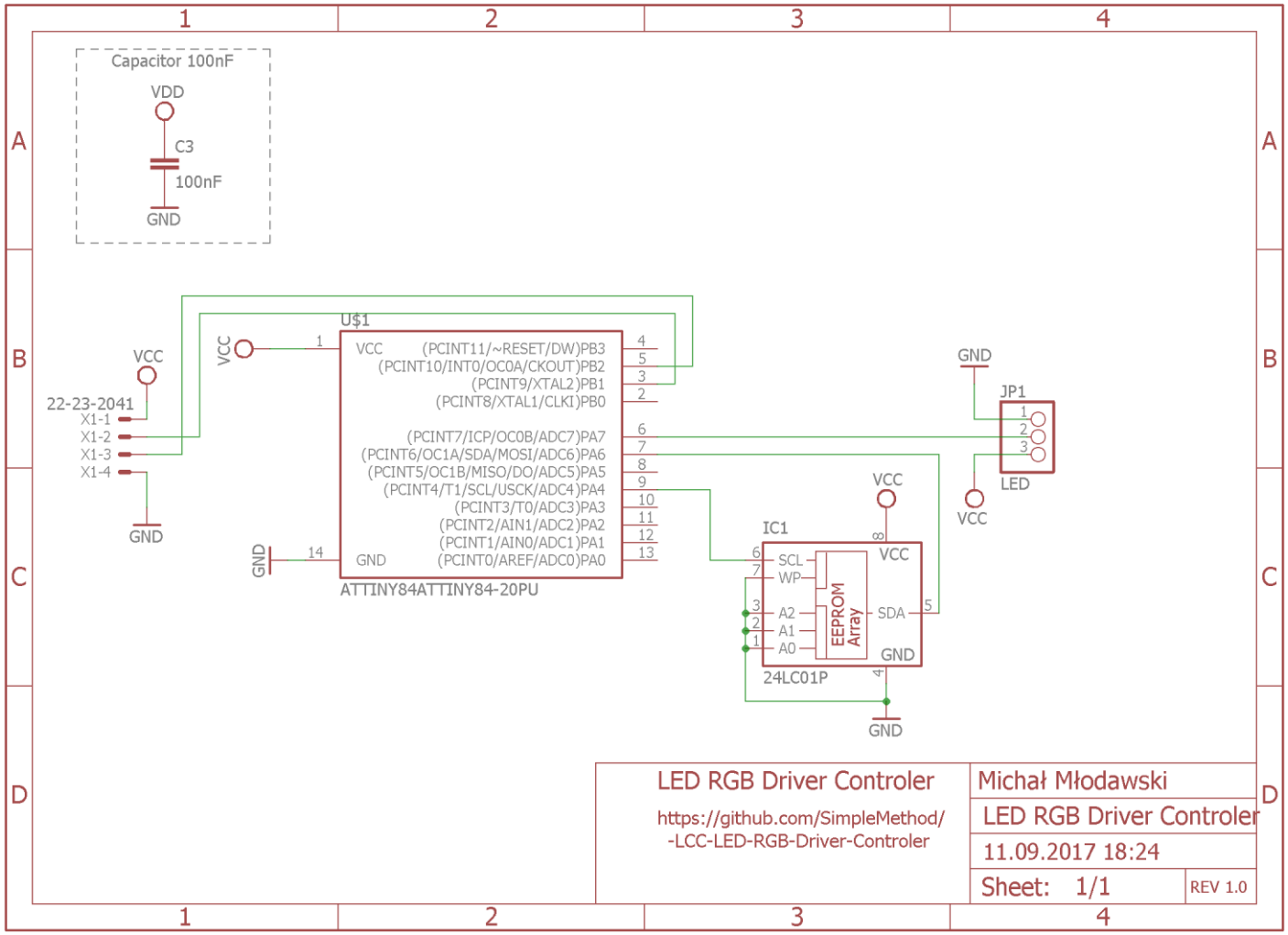
6. Budowa poszczególnych modułów

1. Schemat budowy mostka USB



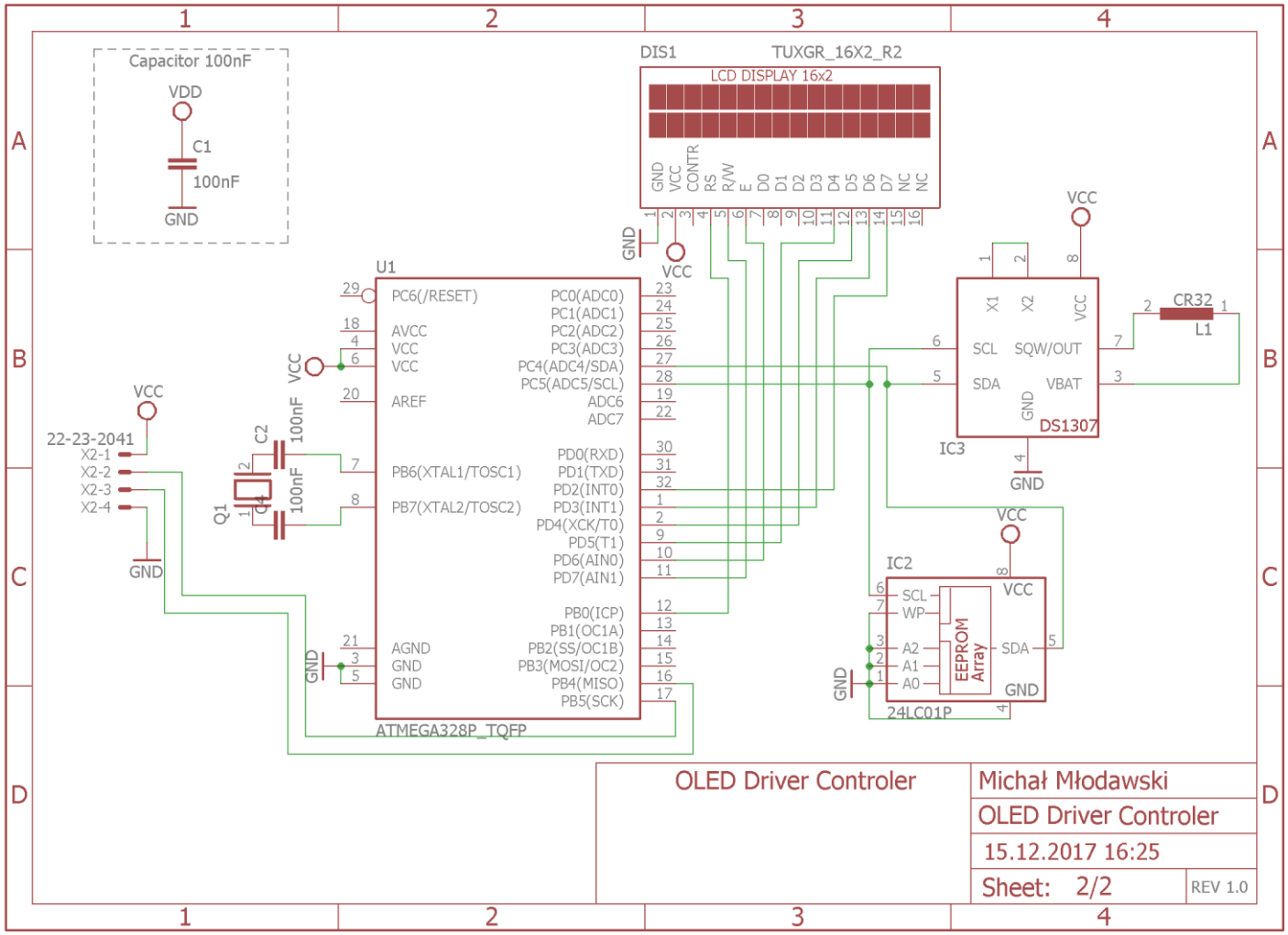
Rysunek 5 Schemat budowy mostka USB wygenerowany w programie Eagle

2. Schemat budowy sterownika LED



Rysunek 6 Schemat budowy sterownika LED wygenerowany w programie Eagle.

3. Schemat budowy sterownika OLED



Rysunek 7Schemat budowy sterownika OLED wygenerowany w programie Eagle (Uproszczony)

7. Mikrokod dla modułów LED oraz OLED

1. Mikrokod dla modułu LED

1. Mikrokod rozkazu dla modułu LED

LP.	Pierwszy człon rozkazu:	Drugi człon rozkazu:	Przyjmowany typ danych:	Opis rozkazu:	Uwagi:
1.	0000	001	Byte	Wyświetlanie tekstu	Brak
2.	0000	002	Long	Wysyła zawartość komórki	Wymagane połączenie z użyciem RS-232
3.	0000	003	Typ złożony	Aktualizacja pamięci EEPROM	Pierwsze 3 znaki odpowiadają wartości, kolejne 17 adresowi komórki

Tabela 2 Mikrokod rozkazu dla modułu LED.

2. Mikrokod odpowiedzi modułu LED

LP.	Pierwszy człon rozkazu:	Drugi człon rozkazu:	Wartość trzeciego członu rozkazu:	Typ danych:	Opis odpowiedzi:	Uwagi:
1.	0001	003	255	Byte	Zwraca pomyślne uaktualnianie pamięci.	Brak
2.	0001	003	000	Byte	Zwraca nieudane uaktualnianie pamięci.	Brak

Tabela 3 Mikrokod odpowiedzi modułu LED.

2. Mikrokod dla modułu OLED

1. Mikrokod rozkazu dla modułu OLED

LP.	Pierwszy człon rozkazu:	Drugi człon rozkazu:	Przyjmowany typ danych:	Opis rozkazu:	Uwagi:
1.	0000	001	String	Wyświetlanie tekstu na ekranie OLED	Maksymalnie 41 znaków
2.	0000	002	String	Wyświetlanie tekstu w formie jego przesuwania na ekranie OLED.	Maksymalnie 41 znaków.
3.	0000	003	Typ złożony	Aktualizacja pamięci EEPROM.	Pierwsze 3 znaki odpowiadają wartości, kolejne 17 adresowi komórki.
4.	0000	004	Brak	Wyświetla podstawowy bufor pamięci grafiki.	Brak
5.	0000	005	Brak	Wysyła sygnał wybudzenia do kontrolera.	Brak
6.	0000	006	Long	Wyświetla na ekranie zawartość komórki.	Brak
7.	0000	007	Long	Wyświetla podstawowy bufor pamięci grafiki zaczynając od podanego bitu.	Brak
8.	0000	008	Brak	Aktualizuje zegar RTC na	Zwraca komunikat o

				podstawie danych z pamięci EEPROM.	udanej aktualizacji .
9.	0000	009	Brak	Wyświetla na ekranie aktualny dzień i godzinę.	Brak
10.	0000	010	Brak	Ustawia flagę zegara sektorze pamięci BootLoader.	Brak
11.	0000	011	Brak	Zeruje flagę zegara sektorze pamięci BootLoader.	Brak
12.	000	012	Brak	Zwraca całą zwartość pamięci EEPROM.	Wymagane połączenie z użyciem RS-232.

Tabela 4 Mikrokod rozkazu dla modułu OLED

2. Mikrokod odpowiedzi modułu OLED

LP.	Pierwszy człon rozkazu:	Drugi człon rozkazu:	Wartość trzeciego członu rozkazu:	Typ danych:	Opis odpowiedzi:	Uwagi:
1.	0001	003	255	Byte	Zwraca pomyślne uaktualnianie pamięci.	Brak
2.	0001	003	000	Byte	Zwraca nieudane uaktualnianie pamięci.	Brak
3.	0001	005	255	Byte	Sygnał wybudzenia.	Brak
4.	0001	008	255	Brak	Zwraca pomyślne uaktualnianie pamięci zegara RTC.	Brak

Tabela 5 Mikrokod odpowiedzi modułu OLED

8. Opis poszczególnych klas i metod

1. Opis klas

1. Klasa BackEnd.h zajmuje się Back-Endem aplikacji. Jest to klasa zajmująca się obsługą aplikacji od strony technicznej. Znajdują się w niej główne metody, dzięki którym można stworzyć plik, czy połączyć się z portem COM.
2. Klasa FrontEnd.h zajmuje się Front-Endem aplikacji. Jest to klasa zajmująca się obsługą aplikacji od strony wizualnej. Znajdują się w niej główne metody, dzięki którym można komunikować się z JavaScript.

2. Opis metod

1. Klasa BackEnd.h

LP.	Metoda:	Argument przyjmujący:	Argument zwracany:	Opis:
1.	bool CreateFile(QString file);	QString file-Nazwa pliku jaki należy stworzyć.	Bool-Zwraca prawdę albo fałsz	Metoda tworząca dowolny plik.
2.	bool CheckFileExist(QString file);	QString file-Nazwa pliku jaki należy stworzyć.	Bool-Zwraca prawdę albo fałsz	Metoda sprawdzająca, czy plik istnieje.
3.	bool CheckDLCisInstalled(AppId_t ID);	AppId_t ID- Numer SteamID	Bool- Zwraca prawdę albo fałsz.	Metoda sprawdzająca, czy DLC jest zainstalowane.
4.	QString SteamLang();	Brak	QString Zwraca tekst do przetłumaczenia.	Metoda sprawdzająca, jaki język został ustawiony.
5.	void SelectLanguage(QString LanguageName);	QString LanguageName - Mazwa języka jaki został ustawiony.	QString Zwraca identyfikator dla odpowiedniego języka.	Metoda przypisuje do zmiennej RawJsonData całą zawartość pliku Json z odpowiednimi plikami językowymi.
6.	void SelectSettingFile(QString LanguageName);	QString filename - Nazwa pliku językowego.	Brak	Metoda przypisuje do zmiennej RawJsonData całą zawartość pliku Json z odpowiednimi plikami językowymi.
7.	QJsonDocument OpenJsonFile(QString filename);	QString filename - Nazwa pliku językowego.	QJsonDocument Zwraca dump pliku językowego.	Metoda otwiera plik Json.
9.	QString SelectFromJson(QString SearchName)	QString SearchName -	QJsonDocument Zwraca	Metoda czyta plik Json i wybiera element nie

		Nazwa pliku językowego.	odpowiedną wartość z pliku.	znajdujący się w klasie.
10.	QString SelectFromArrayJson(QString NameOfArray, QString SearchName);	QString NameOfArray - Nazwa tablicy w pliku językowego. QString SearchName - Nazwa elemetnu w tablicy pliku językowego.	QJsonDocument Zwraca odpowiedną wartość z pliku.	Metoda czyta plik Json i wybiera element z tablicy znajdujący się w klasie.
11.	QString ScanCOMPort(QString ProductID);	QString ProductID - Nazwa poszukiwanego produktu o podanym indentyfikatorze produktu.	QString-Zwraca nazwę portu COM.	Metoda skanuje wszystkie porty COM.
12.	void InitSendMessageToPort(QString ComPort, QString message);	QString ComPort- Nazwa portu COM.	Brak	Metoda inicjalizuje port COM (Otwiera go).
13.	void SendMessageToPort(QString message);	QString ComPort- Nazwa portu COM QString message wiadomość do przesyłu.	Brak	Metoda Pozwalająca na przesłanie wiadomości do portu COM.

Tabela 6 Opis metod w klasie BackEnd.h

2. Klasa FrontEnd.h

LP.	**Metoda:**	**Argument przyjmujący:**	**Argument zwracany:**	**Opis:**
1.	QString setsteamnick();	Brak	QString-Zwraca nick steam.	Metoda ustawia nick w JavaScript.
2.	QString setslangtext();	Brak	QString-Zwraca polę w którym jest tekst do tłumaczenia.	Metoda ustawia pole językowe w JavaScript.
3.	QString setslcomportname();	Brak	QString-Zwraca nazwę portu COM.	Metoda sprawdzająca, czy DLC jest zainstalowane

4.	bool checkdlc();	Brak	Bool-Zwraca prawdę jeśli DLC jest zainstalowane.	Metoda ustawia DLC w JavaScript.
5.	void initialization_QML();	Brak	Brak	Metoda inicjalizuje JavaScript poprzez załadowanie pliku językowego.
6.	void initialization_QML_MAIN();	Brak	Brak	Metoda zarządza stroną główną, sprawdza, czy porty COM są dostępne oraz tłumaczy ją na określony język.
7.	void initialization_QML_RIGHTBAR();	Brak	Brak	Metoda zarządza prawym paskiem, sprawdza, czy DLC są zainstalowane oraz tłumaczy ją na określony język.
9.	void initialization_QML_LED();	Brak	QJsonDocument - Zwraca odpowiedną wartość z pliku.	Metoda zarządza stroną LED tłumaczy ją na określony język.
10.	void initialization_QML_OLED();	Brak	QJsonDocument - Zwraca odpowiedną wartość z pliku.	Metoda zarządza stroną OLLED tłumaczy ją na określony język.

Tabela 7 Opis metod w klasie FrontEnd.h

3. Opis sygnałów dla FrontEnd.h

LP.	Sygnał:	Argument przyjmujący:	Argument zwracany:	Opis:
1.	void sendSteamNick(int count);	int count -Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej count do JavaScript i zapisuje Steam nick.
2.	void sendDLCInstalled(int count);	int count -Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej count do JavaScript i jest aktywne jeśli DLC jest zainstalowane.
3.	void sendLangTextLed(int signal_led);	int signal_led-Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej signal_led do JavaScript. Ten sygnał zajmuje się tłumaczeniem i zarządzaniem strony LED.
4.	void sendLangTextRightSidebar(int signal_rightsidebar);	int signal_rightsidebar - Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej signal_rightsidebar do JavaScript. Ten sygnał zajmuje się tłumaczeniem i zarządzaniem prawym paskiem.
5.	void sendCOMPort(int signal_mainpage_comport_int);	int signal_mainpage_comport_int- Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej signal_mainpage_comport_int do JavaScript. Ten sygnał zajmuje się tłumaczeniem i zarządzaniem głównym oknem. Dodatkowo zajmuje się przypisaniem portów COM.
6.	void sendLangTextOled(int signal_oled);	int signal_oled- Numer rozkazu do wykonania.	Brak	Sygnał wysyła rozkaz zwarty w zmiennej signal_oled do JavaScript. Ten sygnał zajmuje się tłumaczeniem i zarządzaniem okien OLED.

Tabela 8 Opis sygnałów w klasie FrontEnd.

4. Opis public slots dla FrontEnd.h

LP.	Opis public slots:	Argument przyjmujący:	Argument zwracany:	Opis:
1.	Void receiveFromOledPage(int option,QString OLED);	int option- Zwraca numer rozkazu do wykonania. QString OLED Zwraca zawartość rozkazu do wykonania.	Brak	Slots zajmuje się odebraniem danych ze strony OLED JavaScript i przekazaniem ich do C++.
2.	void receiveFromLedPage (int option,int value);	int option- Zwraca numer rozkazu do wykonania. int value Zwraca zawartość rozkazu do wykonania.	Brak	Slots zajmuje się odebraniem danych ze strony LED JavaScript i przekazaniem ich do C++.

Tabela 9 Opis public slots dla FrontEnd.h

5. Opis funkcji dla modułu LED

LP.	Funkcja:	Argument przyjmujący:	Argument zwracany:	Opis:
1.	void setup();	Brak	Brak	Główna funkcja uruchomiona w momencie startu modułu, służy do zainicjowania komunikacji między modułem, a mostkiem USB, a także do ustawienie częstotliwości taktowania.
2.	void loop();	Brak	Brak	funkcja, która się wykonuje z każdym cyklem procesora.
3.	void writeAddress(int address, byte val);	int address – Adres komórki do jakiej zapiszemy byte val – Wartość jaką zapiszemy w pamięci	Brak	Funkcja za pomocą I²C przesyła do pamięci EEPROM wartość w określonej komórce i tam ją zapisuje.
4.	byte readAddress(int address);	int address – Adres komórki, którą chcemy odczytać	byte -Zwraca wartość komórki pamięci	Funkcja za pomocą I²C przesyła do pamięci EEPROM

				żądanie odczytania danej komórki i pobiera z niej dane.
5.	void ReadCommand(String Text, byte Def);	String Text- Rozkazu w postaci tekstu do wykonania Def podstawowy rozkaz do wykonania Byte Def- podstawowa funkcja	Brak	Funkcja, która ustawia LED i pozwala aktualizować pamięć EEPROM. Mikrokod dla ten funkcji znajduje się w tabeli 2

6. Opis funkcji dla modułu OLED

LP.	Funkcja:	Argument przyjmujący:	Argument zwracany:	Opis:
1.	void setup();	Brak	Brak	Główna funkcja uruchomiona w momencie startu modułu, służy do zainicjowania komunikacji między modułem, a mostkiem USB, a także do ustawienie częstotliwości taktowania.
2.	void loop();	Brak	Brak	funkcja, która się wykonuje z każdym cyklem procesora.
3.	void writeAddress(int address, byte val);	int address – Adres komórki do jakiej zapiszemy byte val – Wartość jaką zapiszemy w pamięci	Brak	Funkcja za pomocą I²C przesyła do pamięci EEPROM wartość w określonej komórce i tam ją zapisuje.
4.	byte readAddress(int address);	int address – Adres komórki, którą chcemy odczytać	byte -Zwraca wartość komórki pamięci	Funkcja za pomocą I²C przesyła do

				pamięci EEPROM żądanie odczytania danej komórki i pobiera z niej dane.
5.	void ReadCommand(String Text, byte Def);	String Text- Rozkazu w postaci tekstu do wykonania Def podstawowy rozkaz do wykonania Byte Def- podstawowa funkcja	Brak	Funkcja, która ustawia OLED i pozwala aktualizować pamięć EEPROM Mikrokod dla ten funkcji znajduje się w tabeli 4
6.	void ScrollingMarquee (String u1);	String u1- Tekst do przesunięcia.	Brak	Funkcja ułatwia przesuwanie tekstu z lewej do prawej na ekranie OLED.
7.	void DisplayGraphic (int Data);	int Data -Adres pamięci od którego wyświetlacz ma czytać pamięć EEPROM.	Brak	Funkcja, która ustawia wyświetlacz w trybie Graficznym.
8.	void DisplayTime();	Brak	Brak	Funkcja, która wyświetla dzień i godzinę na ekranie.

9. Podsumowanie projektu

Podczas prac nad projektem natknęliśmy się na kilka problemów, takich jak trudność połączeniem JavaScript z C++, czy odpowiednią synchronizację wyświetlacza OLED z mikrokontrolerem Atmega 328p. Pracowaliśmy razem nad większością elementów projektu. Michał Mołdawski przygotował schematy budowy modułów, BackEnd aplikacji, częściowo FrontEnd, oprogramowanie do modułu OLED oraz pliki JSON. Konrad Nowakowski opracował oprogramowanie do modułu LED, częściowo FrontEnd aplikacji i częściowo kod JavaScript, przygotował dokumentację w DoxyGen i wspomógł pracę przy opracowaniu . Wspólnie opracowaliśmy kod w JavaScript, dokumentację projektu, a także fizycznie wytworzyliśmy moduły. Chciałbym podziękować firmie Transfer Multisort Elektronik za wsparcie i dostarczenie elementów do modułów oraz społeczności StackOverflow dzięki której rozwiązaliśmy problem synchronizacji wyświetlacza.