Datalogger - Dokumentacja

Revision: 1.0

Spis treści

1 Instrukcja obsługi

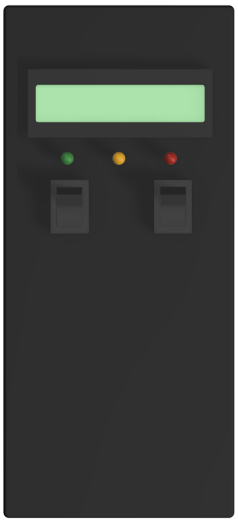
1.1 Dane techniczne

- **Zakres napięcia zasilającego**: 11-16V DC
- **Rekomendowany sposób zasilania**: Pakiet bateryjny 3S
- **Średni pobór mocy**: 2W
- **Rezystancja wejściowa pomiaru cyfrowego**: 1.2kΩ
- **Napięcie wejściowe pomiaru cyfrowego**: 24V
- **Zakres napięcia pomiaru analogowego**: 0-10V
- **Maksymalna oporność rezystora pętli prądowej**: 450Ω
- **Złącza przyłączeniowe**: Złącza śrubowe

- **Kompatybilny typ tagów RFID**: 13,6MHz
- **Minimalna odległość zbliżenia tagu RFID**: 3cm

- **Wymiary**: 200,2x90,1x49 mm
- **Masa**: ok. 150g (bez baterii)

1.2 Ułożenie elementów



(a) Przód urządzenia z wyświetlaczem, diodami sygnalizującymi stan urządzenia oraz włącznikami.

(b) Przód urządzenia ze złączami - od lewej pomiar cyfrowy, pomiar analogowy, wyjście 0-10V, wyjście 4..20mA.

Rysunek 1: Rozmieszczenie elementów wewnątrz urządzenia.

Patrząc od przodu urządzenia od lewej znajduje się:

1. Wejście GND pomiaru cyfrowego,
2. Wejście OK pomiaru cyfrowego,
3. Wejście NOK pomiaru cyfrowego,
4. Wejście GND pomiaru analogowego,
5. Wejście + pomiaru analogowego,
6. Wyjście GND 0-10V,
7. Wyjście + 0-10V,
8. Wyjście - 4..20mA,
9. Wyjście + 4..20mA.

1.3 Instalacja urządzenia

1. Urządzenie zamontować z dala od niekorzystnych warunków takich jak wilgoć, możliwość zalania, kurz i pył.

2. Sprawdzić czy pakiet baterii zasilający urządzenie posiada odpowiednie napięcie. W przypadku kiedy bateria posiada napięcie mniejsze niż 10,8V skorzystać z wejścia do ładowania baterii oznaczonego symbolem J1.

3. Zaleca się stosowanie jak najkrótszych przewodów oraz w razie możliwość ekranowanych.

1.4 Konfiguracja

Przed pierwszym uruchomieniem należy odpowiednio przygotować plik do zapisu danych na karcie SD. Oprogramowanie nie jest w stanie powiększyć rozmiaru pliku zapisanego na karcie co może spowodować, że przy długim czasie pracy urządzenia dane dalej nie będą zapisywane. Domyślna nazwa pliku to "data.txt". Plik powinien mieć dużą liczbę białych znaków tj. spacji. Zaleca się, aby taki plik posiadał co najmniej 100k białych znaków. Karta powinna być uprzednio sformatowana w systemie plików FAT32.

1.5 Funkcjonalność

Aby uruchomić urządzenie należy przełączyć przycisk znajdujący się po lewej stronie urządzenia. Uruchomienie urządzenia sygnalizowane jest poprzez zapalenie wszystkich widocznych diod oraz wyświetlenie komunikatu o aktywacji na wyświetlaczu LCD. Po krótkim odstępie czasu zielona i czerwona dioda zostają zgaszone.

Jeśli czytnik kart RFID nie zostanie wykryty użytkownik zostanie o tym poinformowany poprzez komunikat na wyświetlaczu oraz zmiane diody żółtej na diodę czerwoną.

Jeśli czytnik kart RFID został wykryty wyświetlony zostanie komunikat o potrzebie zalogowania użytkownika poprzez przyłożenie karty RFID. Użytkownik zostaje poinformowany o sukcesie zalogowania poprzez zmianę żółtej diody na zieloną oraz odpowiedni komunikat na wyświetlaczu.

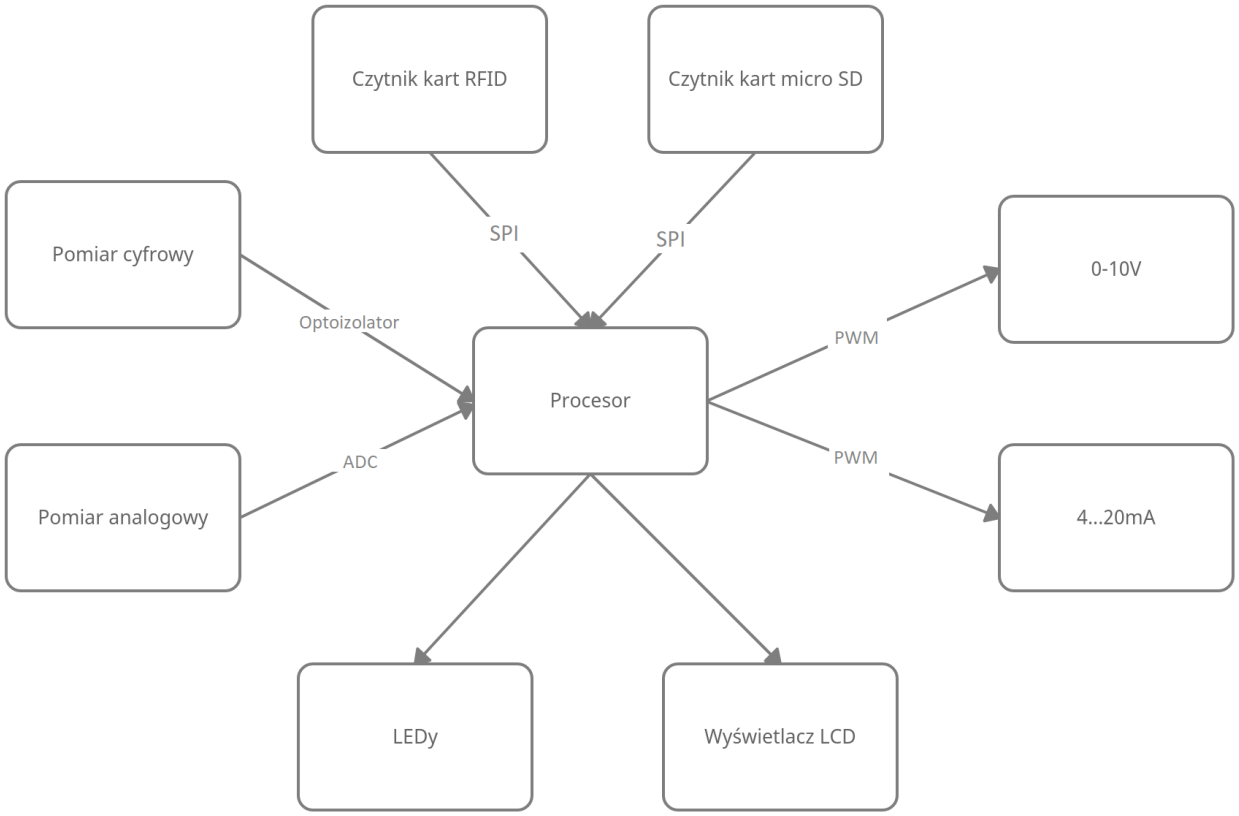
Następnie urządzenie przygotowuje się do zapisu danych na karte SD. Jeśli plik na karcie SD nie zostanie poprawnie otworzony użytkownik otrzyma komunikat podobny do tego w przypadku wystąpienia błędu dla czytnika RFID. Urządzenie kolejno przystępuje do pomiaru ADC w zakresie 0-10V. W zależności od odczytanej wartości zadawany jest odpowiedni prąd (odczytany 1V odpowiada 1,6mA liczonych w zakresie od 4 do 20mA). Kolejno zadawane jest napięcie równe połowie wartości odczytanej

przez pomiar ADC. Po wykonaniu tych trzech sekwencji dane odczytane i zadane zostają zapisane do odpowiedniego bufora. Zakończenie sekwencji sygnalizowane jest poprzez zaświecenie zielonej diody. Jeśli wielkość tego bufora przekroczy dopuszczalną ilość znaków wszystkie dane zostają zapisane na karcie SD. Moment zapisu sygnalizowany jest poprzez zaświecenie żółtej diody. Na koniec urządzenie wraca do pomiaru ADC i cała sekwencja jest powtarzana aż do momentu przerwania przez użytkownika.

Jeśli użytkownik przerwie prace urządzenia zostaje zaświecona czerwona dioda, pojawia się odpowiedni komunikat a dotychczasowe dane zostają zapisane na karcie SD.

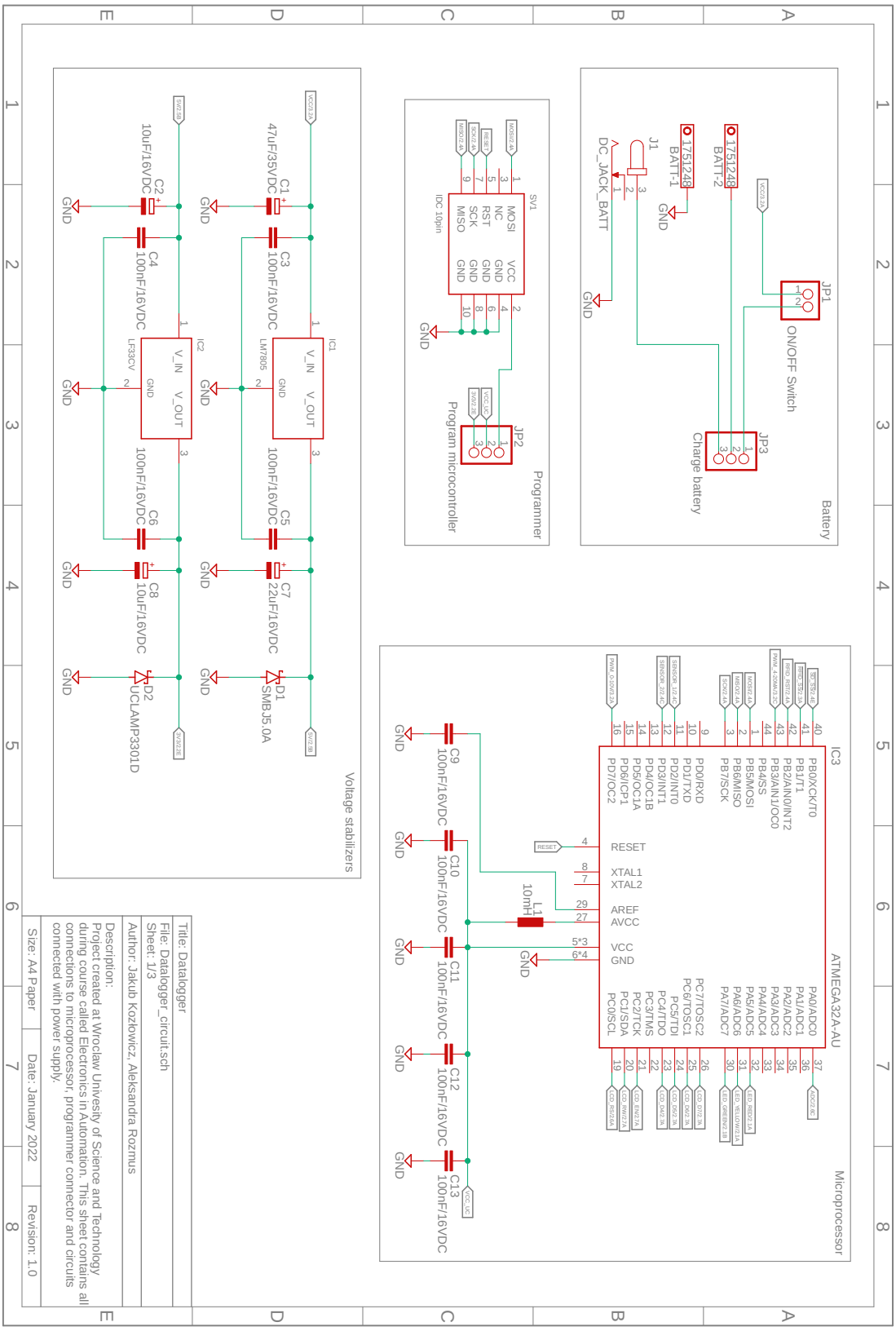
2 Wewnętrzny układ elektryczny

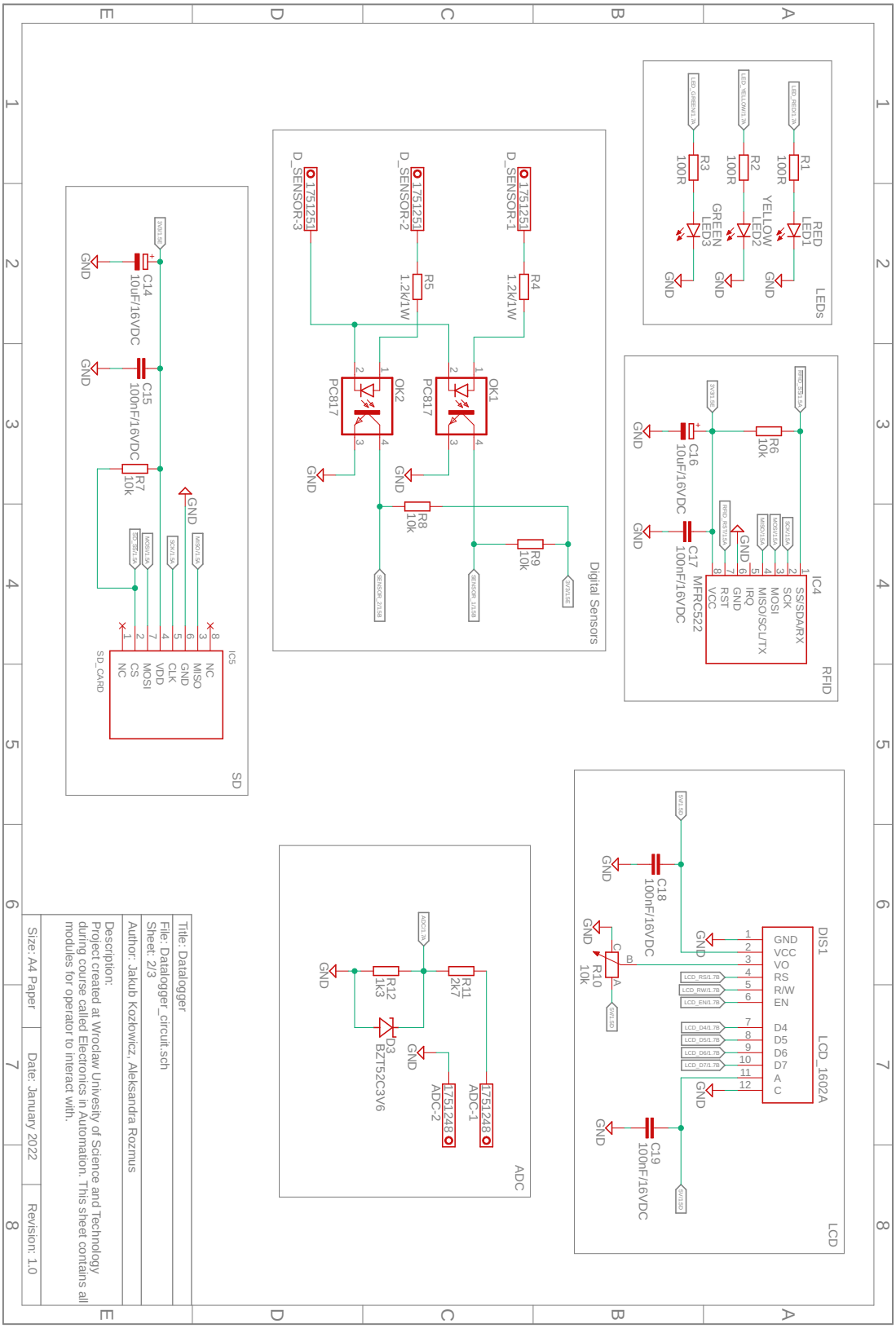
2.1 Schemat blokowy

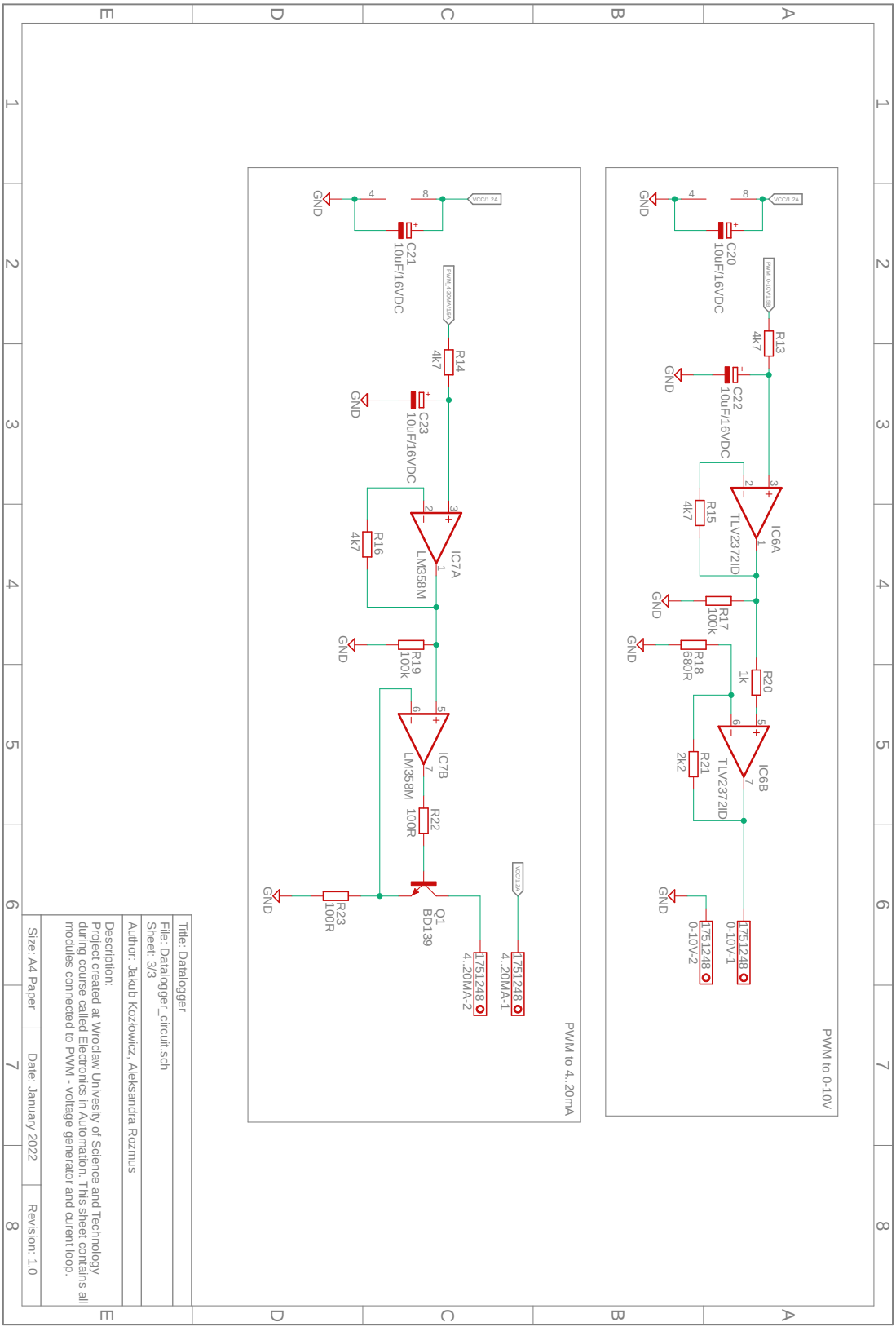


Rysunek 2: Schemat blokowy układu elektronicznego.

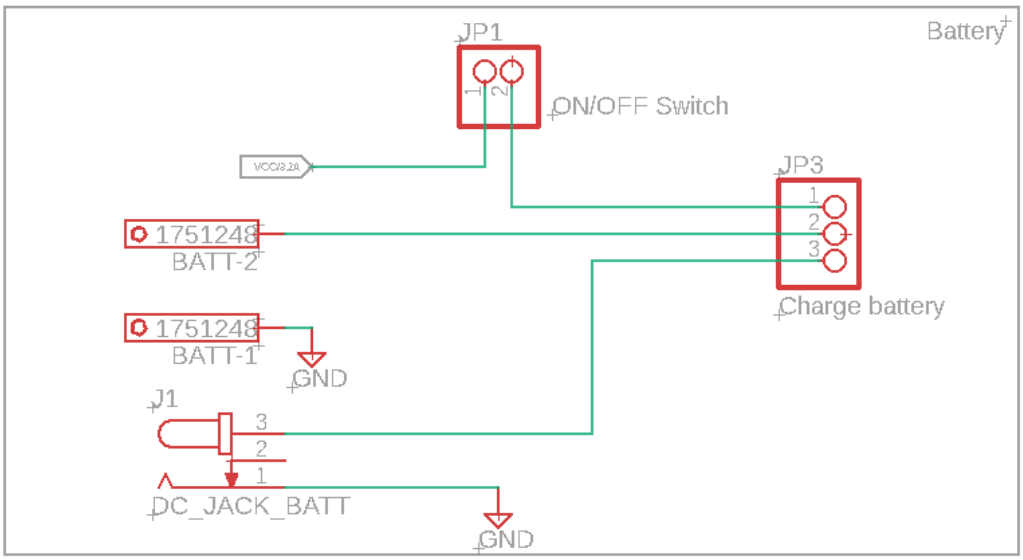
2.2 Schemat ideowy




LEDs
RED
LED1
YELLOW
LED2
GREEN
LED3
R1
100R
R2
100R
R3
100R
GND
RFID
IC4
MFRC522
SS/SDA/RX
SCK
MOSI
MISO/SCL/TX
IRQ
GND
RST
VCC
R6
10k
C16
10uF/16VDC
C17
100nF/16VDC
Digital Sensors
D_SENSOR-1
D_SENSOR-2
D_SENSOR-3
R4
1.2k/1W
R5
1.2k/1W
R8
10k
R9
10k
OK1
OK2
PC817
SD
IC5
SD_CARD
NC
MISO
GND
CLK
VDD
MOSI
CS
NC
C14
10uF/16VDC
C15
100nF/16VDC
R7
10k
LCD
DIS1
LCD_1602A
GND
VCC
VO
RS
R/W
EN
D4
D5
D6
D7
A
C
R10
10k
C18
100nF/16VDC
C19
100nF/16VDC
ADC
ADC-1
ADC-2
R11
2k7
R12
1k3
D3
BZT52C3V6
Title: Datalogger
File: Datalogger_circuit.sch
Sheet: 2/3
Author: Jakub Kozłowicz, Aleksandra Rozmus
Description:
Project created at Wrocław University of Science and Technology during course called Electronics in Automation. This sheet contains all modules for operator to interact with.
Size: A4 Paper
Date: January 2022
Revision: 1.0


PWM to 0-10V
PWM to 4..20mA
C20
10uF/16VDC
C21
10uF/16VDC
C22
10uF/16VDC
C23
10uF/16VDC
R13
4k7
R14
4k7
R15
4k7
R16
4k7
R17
100k
R18
680R
R19
100k
R20
1k
R21
2k2
R22
100R
R23
100R
IC6A
TLV2372ID
IC6B
TLV2372ID
IC7A
LM358M
IC7B
LM358M
Q1
BD139
GND
VCC/1.2A
4..20MA-1
4..20MA-2
0-10V-1
0-10V-2
Title: Datalogger
File: Datalogger_circuit.sch
Sheet: 3/3
Author: Jakub Kozłowicz, Aleksandra Rozmus
Description:
Project created at Wrocław University of Science and Technology during course called Electronics in Automation. This sheet contains all modules connected to PWM - voltage generator and current loop.
Size: A4 Paper
Date: January 2022
Revision: 1.0

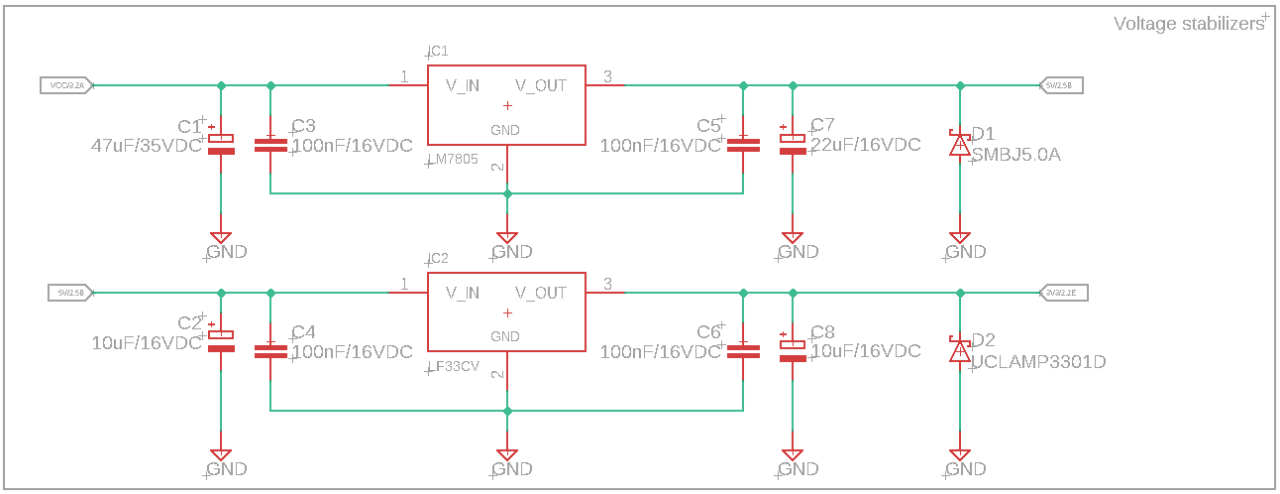
2.3 Układ zasilania



Rysunek 3: Schemat przyłączenia baterii zasilającej urządzenie.

Głównym źródłem zasilania jest pakiet baterii 3S, którego zakres napięć wyjściowych mieści się w zakresie 10,8-12,6V DC. Urządzenie może również być zasilane bezpośrednio z zasilacza napięcia stałego. Jego napięcie wyjściowe powinno mieścić się w zakresie 11-16V DC. Złącze pozwalające podłączyć źródło zasilana to złącze terminalowe DG381-3.5 marki DEGSON ELECTRONICS.

Płytka urządzenia pozwala na bezpośrednie naładowanie pakietu baterii, wyposażonego w układ BMS. Możliwe jest to dzięki zastosowaniu złącza DC o średnicy 2,1/5,5mm oznaczonego symbolem J1 na płytce PCB.

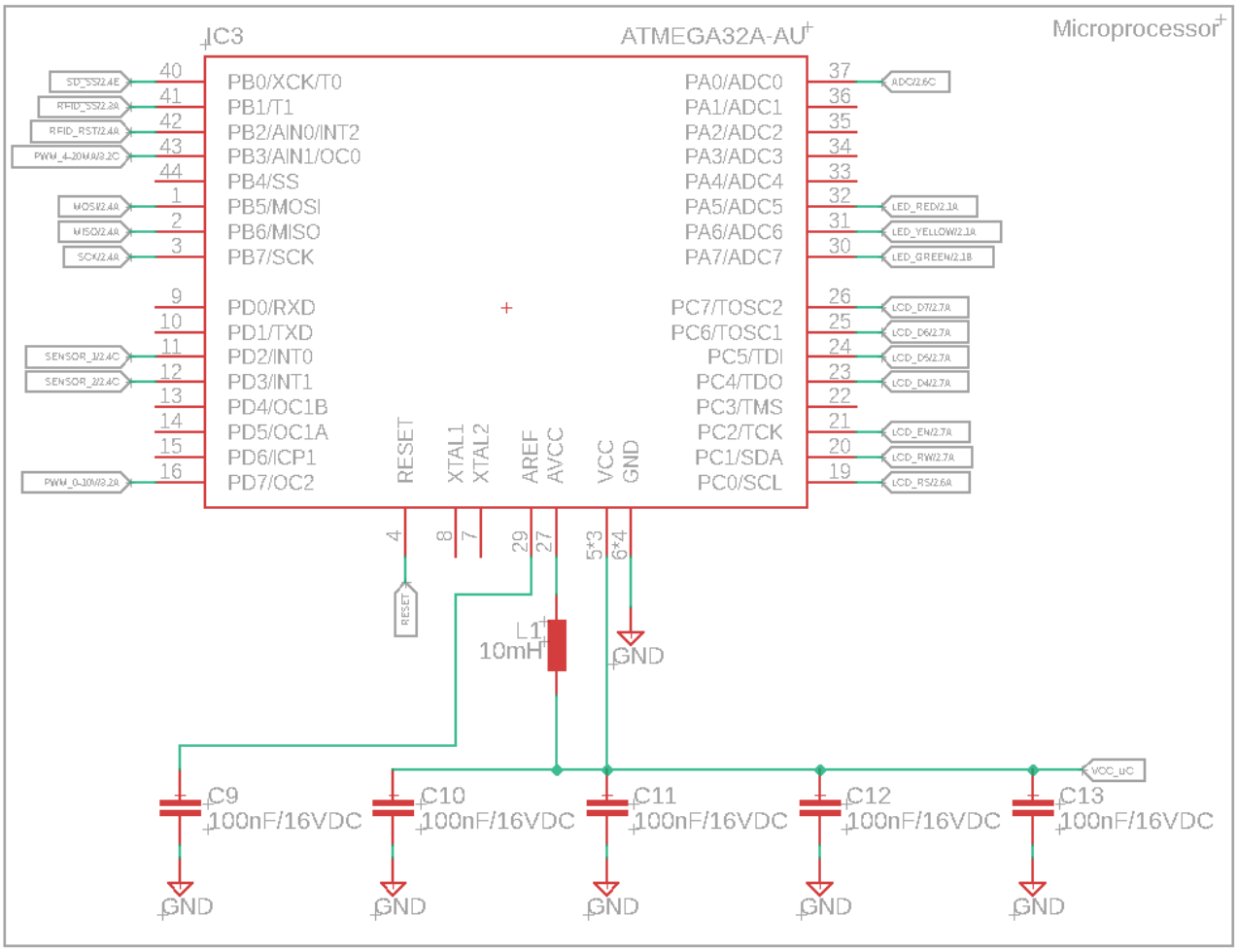


Rysunek 4: Schemat stabilizatorów napięcia zasilających poszczególne moduły.

Jako stabilizatory napięć 5V oraz 3V3 użyto stabilizatorów LM7805 i LF33CV. Regulują one na-

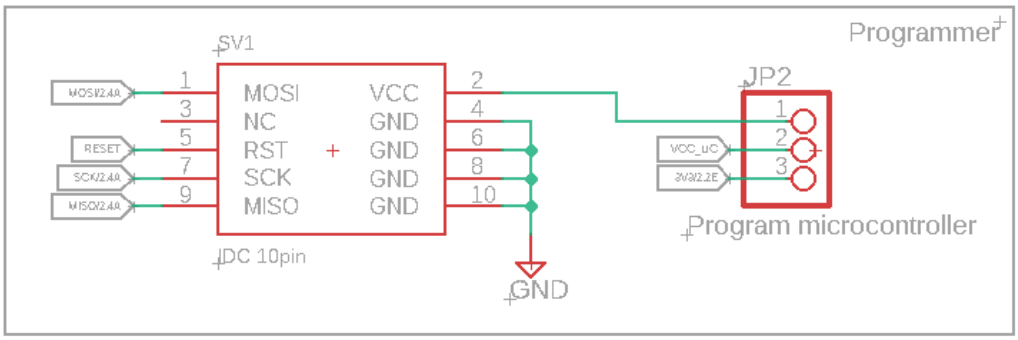
pięcie wejściowe do odpowiednich poziomów. Za zabezpieczenie przeciwprzepięciowe modułów odbiorczych odpowiadają dwie diody transil odpowiednio SMBJ5.0A - 5V, UCLAMP3301D - 3V3. Dioda zabezpieczająca napięcie 5V posiada napięcie przebicia na poziomie 6,4V, a dioda transil dla napięcia 3V3 posiada napięcie przebicia 3V6.

2.4 Procesor



Rysunek 5: Schemat elektroniczny podłączeń do procesora.

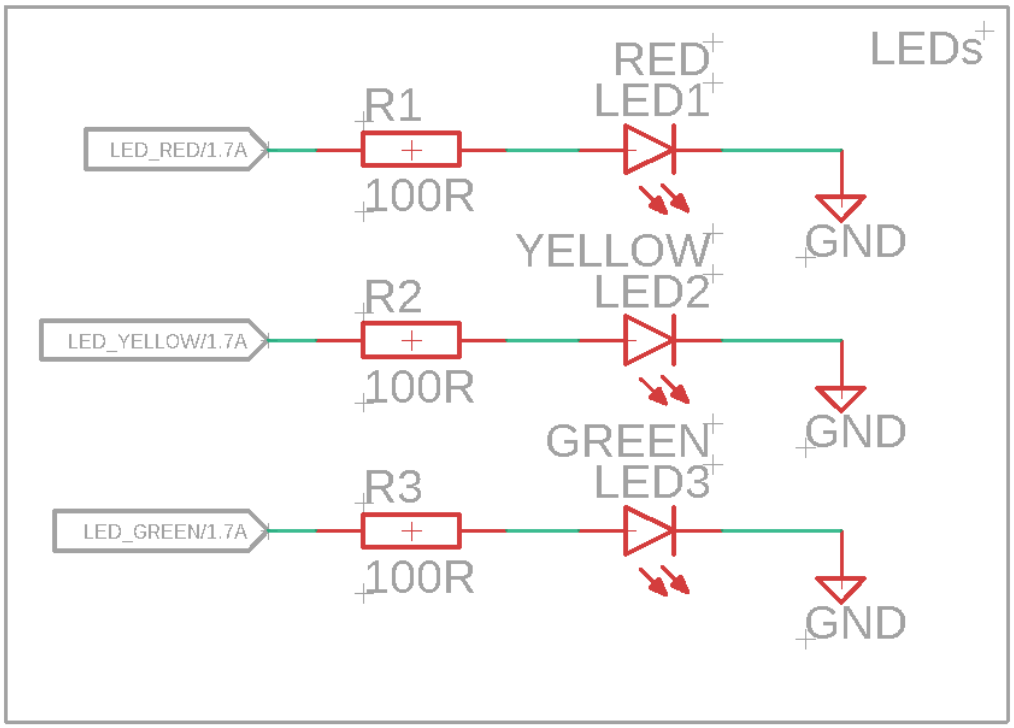
Jako jednostka sterująca wybrany został mikroprocesor Atmega32A w wersji SMD. Komunikuje się on poprzez protokół SPI z dwoma urządzeniami odbiorczymi - modułem czytnika kart RFID oraz kartą micro SD. Steruje on również poprzez sygnał PWM dwoma sygnałami wyjściowymi jakimi są pętla prądowa 4..20mA i zadajnik napięcia 0-10V. Do jednostki sterującej podłączony jest również wyświetlacz LCD 16x2 jak i diody sygnalizujące stan urządzenia. Ostatnim modułem podłączonym do mikroprocesora jest moduł odczytu cyfrowego odizolowany optycznie. Procesor ten pobiera około 15mA.



Rysunek 6: Schemat elektroniczny podłączenia programatora.

Bezpośrednio do procesora podłączone jest złącze serwisowe SV1 do aktualizacji wsadu procesora. Złącze to jest w formacie IDC 10 pin (Kanda). Aby poprawnie zaktualizować procesor należy przełączyć zworkę JP2 w celu zasilenia procesora bezpośrednio z programatora. Komunikuje się on po protokole SPI. Zaleca się wykorzystanie popularnego programatora USBasp.

2.5 Moduł LED



Rysunek 7: Schemat elektroniczny podłączenia diod LED.

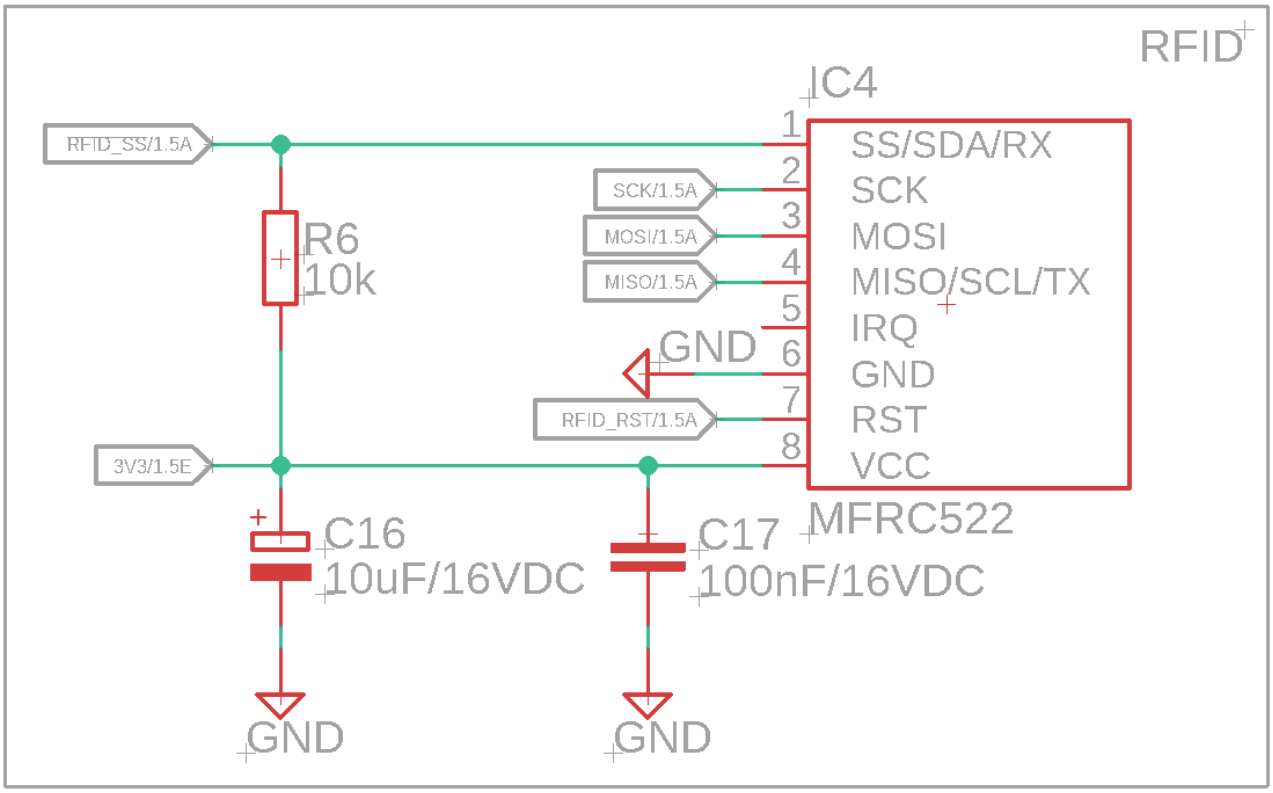
W celu sygnalizacji stanu urządzenia w górnej części obok wyświetlacza LCD zamontowane zostały 3 diody - czerwona, żółta i zielona. Podłączone są one bezpośrednio do procesora poprzez rezystor 100Ω.

W momencie włączenia urządzenia kiedy jest ono gotowe do pracy zaświeci się żółta dioda. Dane przychodzące z pomiaru cyfrowego również są sygnalizowane poprzez diody. Kiedy przyjdzie sygnał, ze dana część została wyprodukowana poprawnie zaświeci się zielona dioda, odwrotnie kiedy sygnał oznacza, że próbka została wyprodukowana niepoprawnie - zaświeci się dioda czerwona. Po każdej poprawnej sekwencji programu zaświeci się dioda żółta co oznacza, że wszystko przeszło bez problemów i program postępuje dalej.

Aby zapewnić poprawną pracę diody dla napięcia 3V3 dodano szeregowo rezystor o wartości 100Ω. Zwiększono wartość rezystancji wychodzącą z obliczeń, aby umożliwić stosowanie większej ilości diod.

$$R = \frac{3,3V - 2V}{20mA} = 65\Omega$$

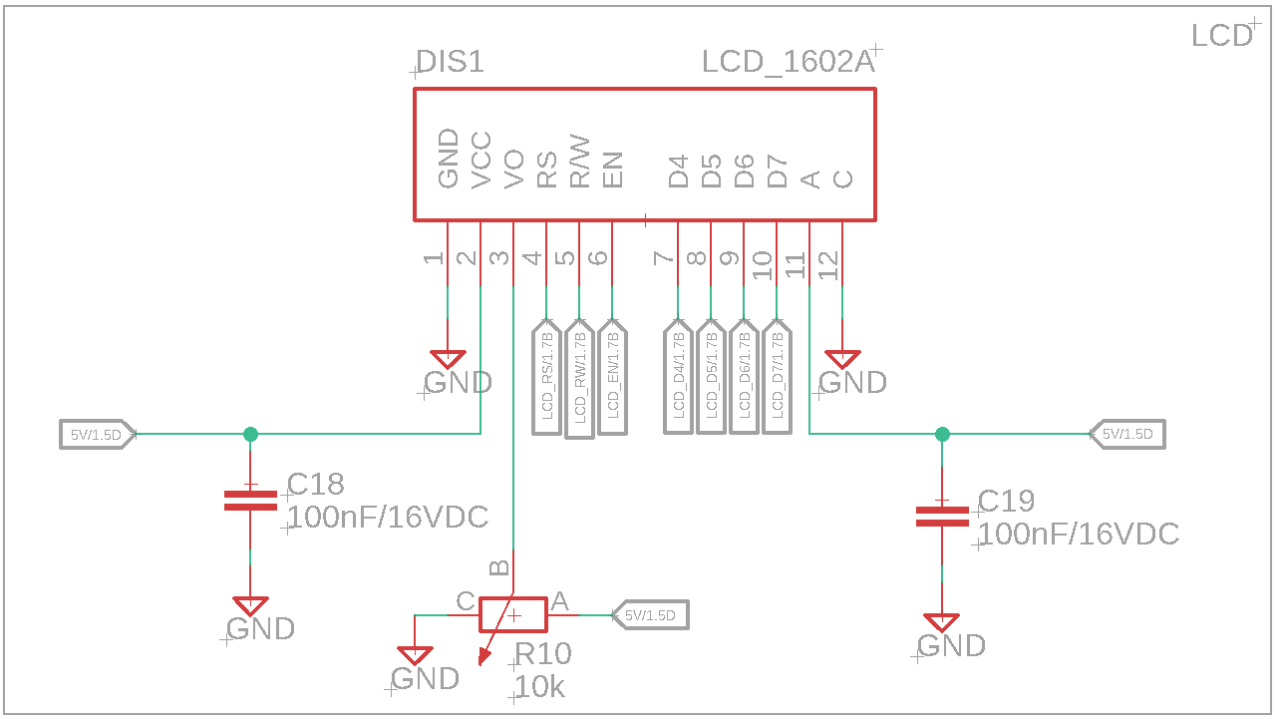
2.6 Moduł RFID



Rysunek 8: Schemat elektroniczny podłączenia czytnika kart RFID.

Za odczyt kart RFID odpowiada moduł czytnika kart RFID MFRC522. Komunikuje się on z procesorem za pomocą magistrali SPI. W celu ograniczenia stanów nieustalonych podczas uruchomiania urządzenia pin Slave Select ($\overline{\text{SS}}$) został podłączony poprzez rezystor 10kΩ do zasilania 3V3. Moduł czytnika kart RFID może pobierać w szczycie nawet 100mA.

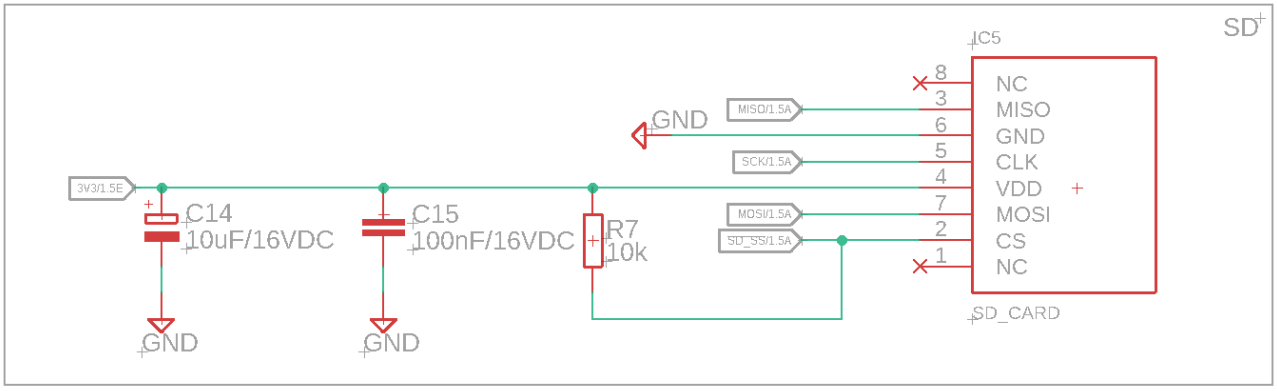
2.7 Moduł LCD



Rysunek 9: Schemat elektroniczny podłączenia wyświetlacza LCD.

Wyświetlacz LCD w konfiguracji 16x2 odpowiada za wyświetlanie komunikatów na temat pracy urządzenia jak i również wartości odczytanych z modułów pomiarowych. Do pinu VO wyświetlacza podłączono potencjometr montażowy 10kΩ. Odpowiada on za poprawne dopasowanie wartości kontrastu wyświetlacza, aby zmaksymalizować jego czytelność.

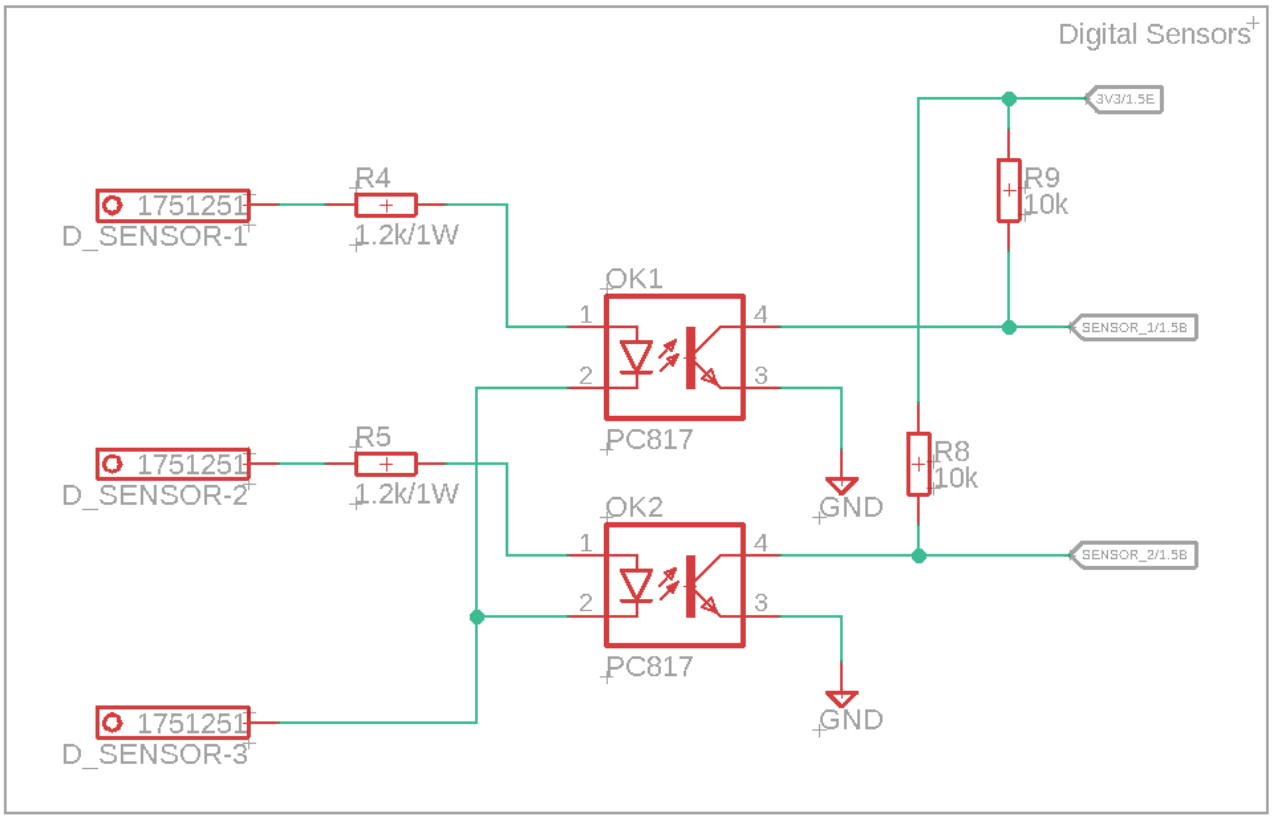
2.8 Moduł SD



Rysunek 10: Schemat elektroniczny podłączenia karty micro SD.

Karta micro SD odpowiada za zachowywanie informacji z pracy urządzenia. Komunikuje się ona poprzez magistralę SPI z mikroprocesorem. Ponownie jak w przypadku czytnika RFID, w celu ograniczenia stanów nieustalonych podczas uruchomienia urządzenia pin Slave Select ($\overline{SS}$) został podłączony poprzez rezystor 10kΩ do zasilania 3V3. W szczycie karta micro SD może pobierać nawet ok. 200mA.

2.9 Moduł pomiaru cyfrowego



Rysunek 11: Schemat elektroniczny podłączenia modułu odczytu cyfrowego z izolacją optyczną.

Za izolacje optyczną wejść cyfrowych odpowiada transoptor PC817. Rezystor mocy 1,2kΩ ogranicza prąd diody tak, aby zapewnić poprawną pracę dla napięcia wejściowego 24V. Złącza wejściowe pomiaru cyfrowego to złącza śrubowe Phoenix Contact 5442251.

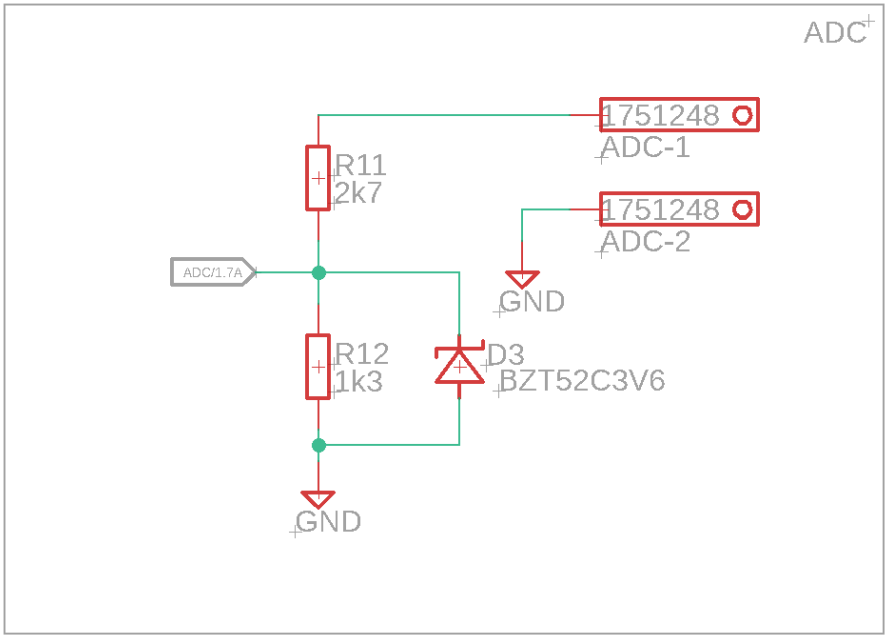
$$U_R = 24V - 1,2V = 22,8V$$

$$R = \frac{U_R}{I} = \frac{22,8V}{20mA} = 1140\Omega \quad \Rightarrow \quad 1,2k\Omega$$

$$P = U_R \cdot I = 22,8V \cdot 20mA = 0,456W$$

Aby zapewnić pewność poprawnej pracy pomiaru cyfrowego wybrano rezystor mocy o wartości maksymalnej 1W.

2.10 Moduł pomiaru analogowego



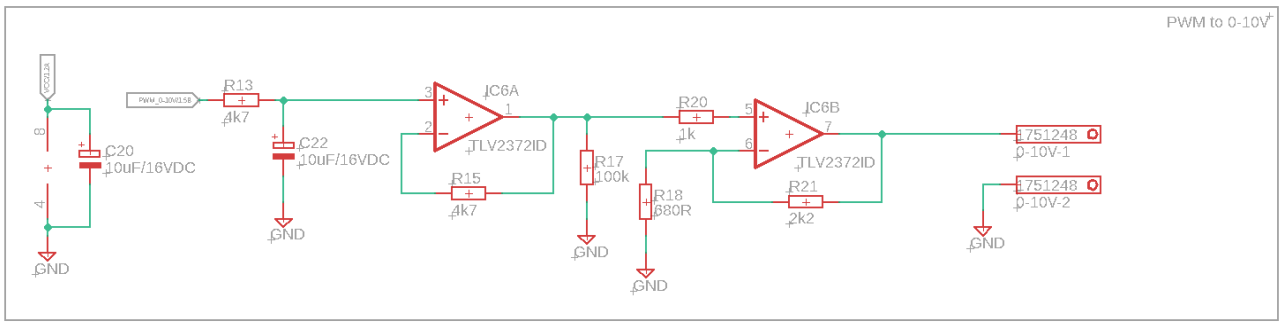
Rysunek 12: Schemat elektroniczny podłączenia modułu odczytu analogowego.

W celu pomiaru analogowego wyprowadzone zostało złącze śrubowe Phoenix Contact 5430001. Zakres napięcia wejściowego pomiaru analogowego mieści się w zakresie 0-10V. W dalszej części zastosowano dzielnik napięcia, w celu ograniczenia poziomu napięcia wchodzącego bezpośrednio na procesor do poziomu 3V3. Dioda Zenera BZT52C3V6 gasi krótkie skoki napięć, aby nie doprowadzić do przepalenia jednostki sterującej. Napięcie Zenera tej diody wynosi 3,6V.

Napięcie na rezystorze R12 można obliczyć ze wzoru

$$V = \frac{V_S \cdot R_{12}}{(R_{11} + R_{12})} = \frac{10V \cdot 1,3k}{2,7k + 1,3k} = 3,25V.$$

2.11 Moduł 0-10V



Rysunek 13: Schemat elektroniczny podłączenia wyjścia napięciowego 0-10V.

Umieszczone na płytce wyjście napięciowe 0-10V sterowane jest sygnałem PWM wychodzącym z procesora. Zastosowany filtr RC przekazuje dalej sygnał na wtórnik napięciowy, który dodatkowo stabilizuje napięcie wchodzące na wzmacniacz nieodwracający o wzmocnieniu 3V/V. W celu ograniczenia stanów nieustalonych wyjście wtórnika napięciowego połączone zostało poprzez rezystor 100kΩ do masy. Zastosowany wzmacniacz to podwójny wzmacniacz operacyjny Rail to Rail - TLV2372ID. Napięcie zasilania tego wzmacniacza to 2,7..16/±1,35..8V DC. Wyprowadzone złącze wyjściowe to złącze śrubowe Phoenix Contact 5430001.

Poprawną funkcjonalność filtru dolnoprzepustowego RC obliczono w sposób następujący

$$f_g = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 4,7k \cdot 10uF} = 3,39Hz,$$

$$\Delta V_{pk-pk} = 0,00056170212223498\ V \quad (\text{Duty} = 50\%),$$

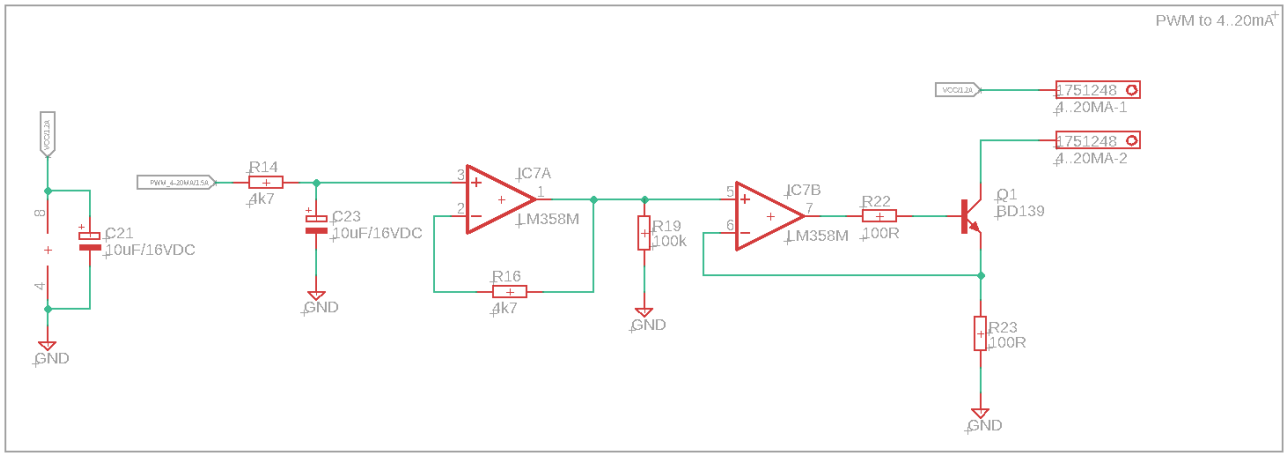
$$tr = 0,10822149937072\ s.$$

Dobranie rezystorów wzmacniacza odwracającego wyglądało następująco

$$K_u = 1 + \frac{R_{21}}{R_{20}} = 3 \quad \Rightarrow \quad \text{zał: } R_{20} = 1\ k\Omega \quad \Rightarrow \quad R_{21} = 2,2\ k\Omega,$$

$$R_{18} = \frac{R_{20} \cdot R_{21}}{R_{20} + R_{21}} = \frac{2,2\ k\Omega}{3,2\ k\Omega} \approx 680\ \Omega.$$

2.12 Moduł 4..20mA



Rysunek 14: Schemat elektroniczny podłączenia pętli prądowej 4..20mA.

Drugim z wyjść sterujących zamontowanych na płytce urządzenia jest pętla prądowa 4..20mA. Sterowana jest ona sygnałem PWM wychodzącym z procesora. Sygnał PWM zamieniany jest na napięcie stałe poprzez ścisłą współpracę filtru RC i wtórnika napięciowego. Ponownie, aby zredukować stany nieustalone podczas rozruchu urządzenia wyjście wtórnika napięciowego zostało podłączone poprzez

rezystor 100kΩ do masy. Pętla prądowa składa się z wzmacniacza operacyjnego LM358M oraz tranzystora NPN - BD139. Złącze do jakiego można podłączyć przewody pętli prądowe to złącza śrubowe Phoenix Contact 5430001. Maksymalna rezystancja opornika pętli prądowej to 450Ω.

Filtr RC został wybrany w sposób identyczny jak w przypadku zadajnika napięciowego 0-10V. Sygnał PWM dla pętli prądowej operuje w zakresie duty 12-60%. Odpowiada to poziomie napięcia 0,4-2V. Tranzystor NPN zapewnia przepływ takiego samego prądu na złączu wyjściowym oraz rezystorze pomiarowym R23. Licząc prąd na rezystorze pomiarowym z prawa Ohma wynika

$$I = \frac{U}{R} = \frac{0,4V}{100\Omega} = 0,004A \quad \Rightarrow \quad I = 4mA,$$

$$I = \frac{U}{R} = \frac{2V}{100\Omega} = 0,02A \quad \Rightarrow \quad I = 20mA.$$

2.13 Tabela elementów

Tablica 1: Tabela oznaczeń elementów i ich wartości.

Oznaczenie na PCB	Wartość
C1	47uF/35VDC
C2	10uF/16VDC
C3	100nF/16VDC
C4	100nF/16VDC
C5	100nF/16VDC
C6	100nF/16VDC
C7	22uF/16VDC
C8	10uF/16VDC
C9	100nF/16VDC
C10	100nF/16VDC
C11	100nF/16VDC
C12	100nF/16VDC
C13	100nF/16VDC
C14	10uF/16VDC
C15	100nF/16VDC
C16	10uF/16VDC
C17	100nF/16VDC

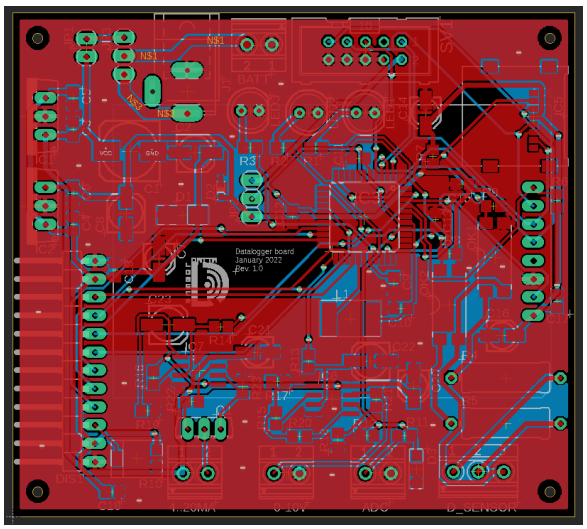
Kontynuacja tabeli 1 z poprzedniej strony.

Oznaczenie na PCB	Wartość
C18	100nF/16VDC
C19	100nF/16VDC
C20	10uF/16VDC
C21	10uF/16VDC
C22	10uF/16VDC
C23	10uF/16VDC
D1	SMBJ5.0A
D2	UCLAMP3301D
D3	BZT52C3V6
DIS1	LCD 1602
IC1	LM7805
IC2	LF33CV
IC3	ATMEGA32A-AU
IC4	MFRC522
IC5	MICRO SD
IC6	TLV2372ID
IC7	LM358M
J1	DC JACK 2,1/5,5
JP1	Włącznik ON/OFF
JP2	3 pin goldpin
JP3	Przełącznik ładowania baterii
L1	10mH
LED1	Czerwony
LED2	Żółty
LED3	Zielony
OK1	PC817
OK2	PC817
R1	100R
R2	100R

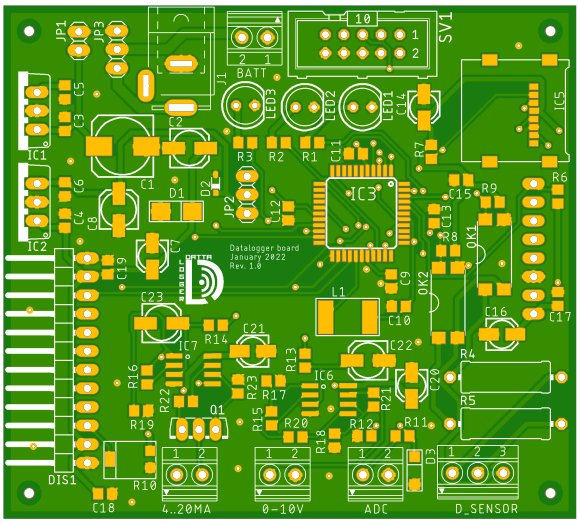
Kontynuacja tabeli 1 z poprzedniej strony.

Oznaczenie na PCB	Wartość
R3	100R
R4	1k2/1W
R5	1k2/1W
R6	10k
R7	10k
R8	10k
R9	10k
R10	Potencjometr 10k
R11	2k7
R12	1k3
R13	4k7
R14	4k7
R15	4k7
R16	4k7
R17	100k
R18	680R
R19	100k
R20	1k
R21	2k2
R22	100R
R23	100R
SV1	IDC 10 pin
Q1	BD139
BATT	ARK 2 pin 3,5mm
D_SENSOR	ARK 3 pin 3,5mm
ADC	ARK 2 pin 3,5mm
0-10V	ARK 2 pin 3,5mm
4..20MA	ARK 2 pin 3,5mm

3 Projekt własnej płytki PCB

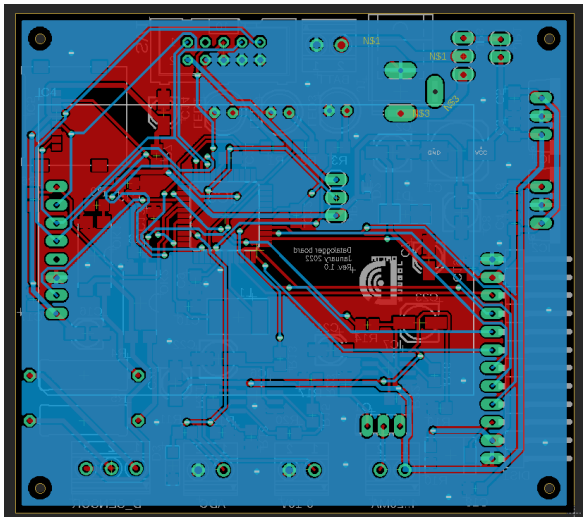


(a) Schemat ścieżek, widok z góry.

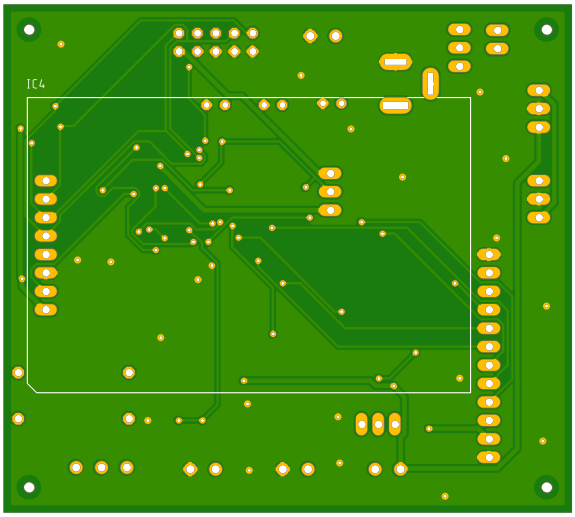


(b) Gotowy widok PCB z góry.

Rysunek 15: Projekt własnej płytki PCB, widok z góry.



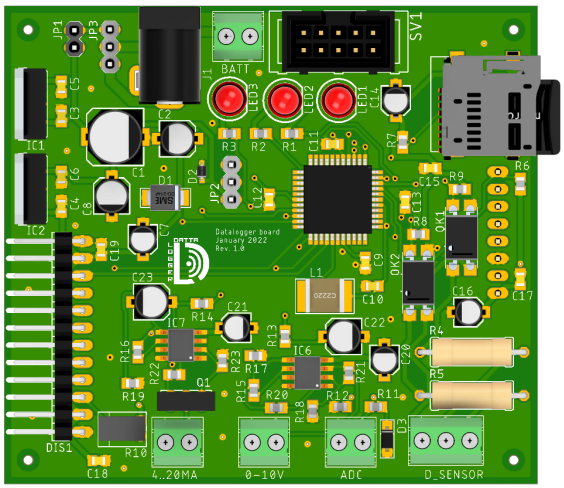
(a) Schemat ścieżek, widok z dołu.



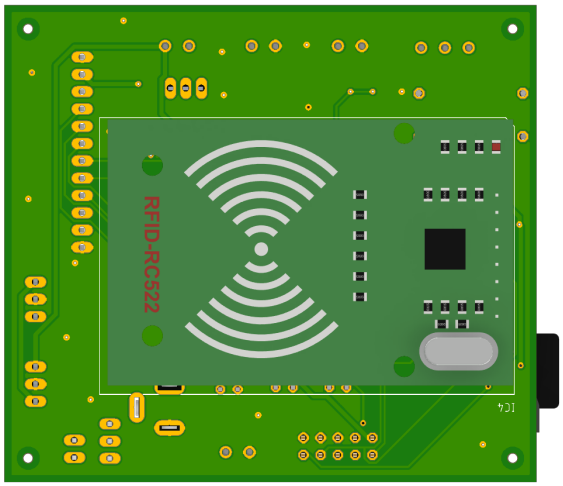
(b) Gotowy widok PCB z dołu.

Rysunek 16: Projekt własnej płytki PCB, widok z dołu.

Układ zaprojektowany został na dwustronnej płytce PCB o wymiarach 70 x 78,2mm. W celu montażu elementów należy zachować standardową kolejność lutowania, tj. od najmniejszych elementów do największych, a na koniec wlutować czytnik kart RFID. Otwory montażowe dostosowane są do uniwersalnej obudowy Kradex Z5B.



(a) Rozmieszczenie elementów na płytce PCB, widok z góry.

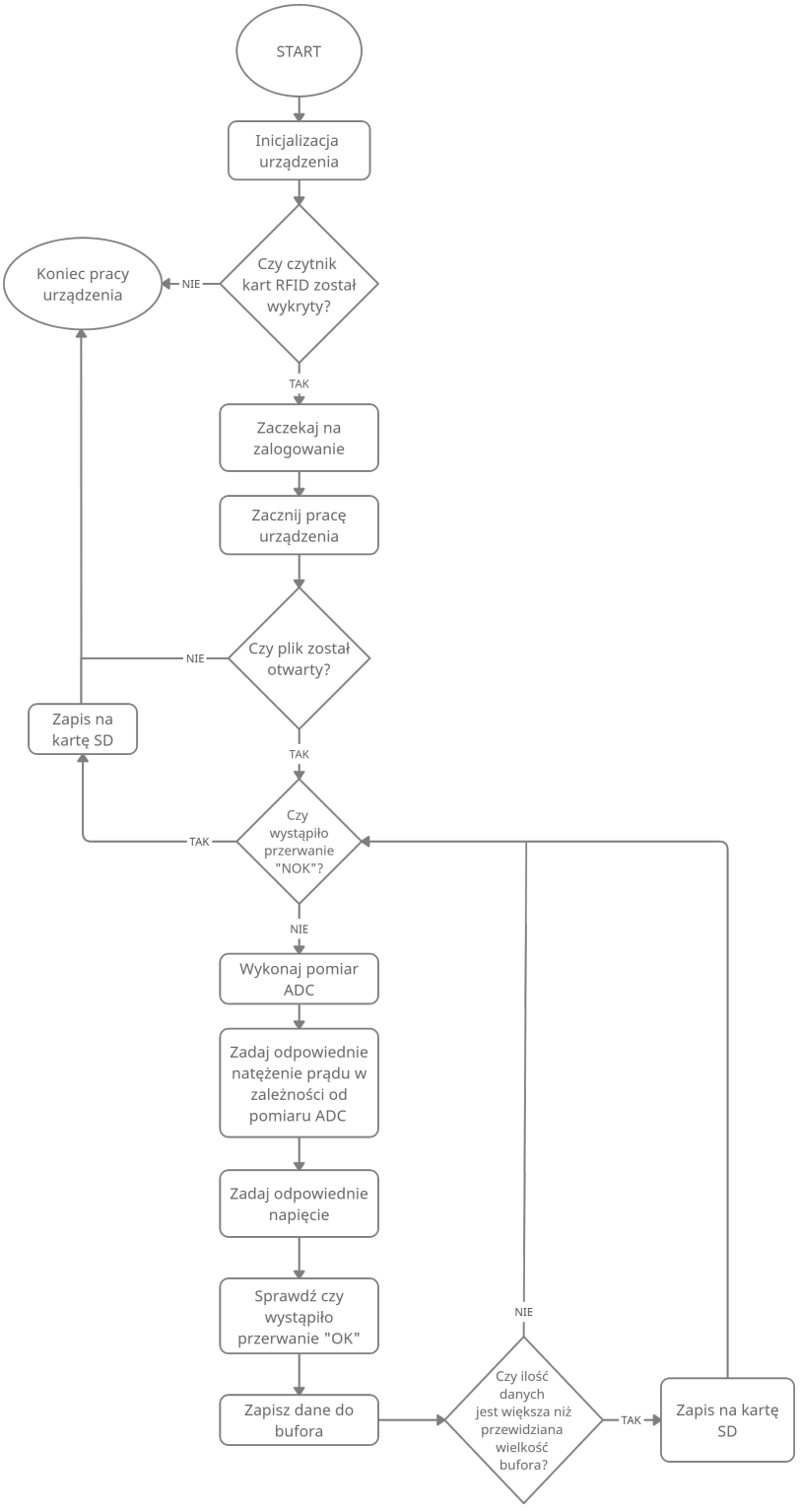


(b) Rozmieszczenie elementów na płytce PCB, widok z dołu.

Rysunek 17: Rozmieszczenie elementów na płytce PCB.

4 Oprogramowanie

4.1 Schemat blokowy



Rysunek 18: Schemat blokowy oprogramowania.

4.2 Tabela sygnałów

Tablica 2: Tabela sygnałów w procesorze.

Lp.		Typ	Opis	Stan: 0	Stan: 1
1	PA0	input	Moduł pomiaru analogowego	-	
2	PA5	output	Czerwona dioda LED	Czerwona dioda zgaszona	Czerwona dioda zapalona
3	PA6	output	Żółta dioda LED	Żółta dioda zgaszona	Żółta dioda zapalona
4	PA7	output	Zielona dioda LED	Zielona dioda zgaszona	Zielona dioda zapalona
5	PB0	output	Slave Select karty SD	Aktywowanie karty SD do transmisji	Dezaktywowanie karty SD do transmisji
6	PB1	output	Slave Select karty RFID	Aktywowanie karty RFID do transmisji	Dezaktywowanie karty RFID do transmisji
7	PB2	output	Linia RESET karty RFID	Resetowanie/wyłączenie modułu	Dezaktywowanie resetu
8	PB3	output	Sterowanie pętlą prądową 4..20mA	Sygnał PWM	
9	PB5	output	Linia MOSI, podłączenie karty SD, karty RFID ora programatora	Magistrala SPI	
10	PB6	input	Linia MISO, podłączenie karty SD, karty RFID oraz programatora		
11	PB7	output	Linia SCK, podłączenie karty SD, karty RFID oraz programatora		
12	PC0	output	Wybór rejestrów wyświetlacza LCD 16x2	Sterowanie wyświetlaczem LCD	
13	PC1	output	Wybór opcji odczyt/zapis wyświetlacza LCD 16x2		
14	PC2	output	Zezwolenie na zapis do rejestrów wyświetlacza LCD 16x2		
15	PC4	output	Pin D4 wyświetlacza LCD 16x2		
16	PC5	output	Pin D5 wyświetlacza LCD 16x2		
17	PC6	output	Pin D6 wyświetlacza LCD 16x2		
18	PC7	output	Pin D7 wyświetlacza LCD 16x2		
19	PD2	input	Moduł pomiaru cyfrowego - OK	Brak reakcji	Zarejestrowanie przerwania OK
20	PD3	input	Moduł pomiaru cyfrowego - NOK	Brak reakcji	Zakończenie pracy urządzenia
21	PD7	output	Sterowanie zadajnikiem napięciowym 0-10V	Sygnał PWM	

4.3 Opis klas programu

4.3.1 ADC

```
/**
 * @file adc.hh
 * @author Jakub Kozłowicz, Aleksandra Rozmus
 * @brief Header file for ADC module
 * @version 0.1
 * @date 2022-01-28
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <avr/io.h>
#include <util/delay.h>

#ifndef ADC_HH_
#define ADC_HH_

class ADC_
{
  private:
    /**
     * @brief Variable to hold the measurement
     *
     */
    double measurement = 0;

  public:
    /**
     * @brief Construct a new ADC_ object
     *
     */
    ADC_();

    /**
     * @brief Measure value of ADC
     *
     * @return uint8_t read value
     */
    double read();

    /**
     * @brief Destroy the ADC_ object
     *
     */
    ~ADC_(){};
};

#endif /* ADC_HH_ */
```

Listing 1: ADC Class.

4.3.2 DISKIO

```
/**
 * @file diskio.hh
 * @author Jakub Kozłowicz, Aleksandra Rozmus
 * @brief Header file for SD Card module. Low level disk interface
    module include file
```

```
 * @version 0.1
 * @date 2022-01-10
 *
 * @copyright Copyright (C) 2009, ChaN
 *
 */

#ifndef _DISKIO

#include "integer.hh"
#include "sd_reg.hh"

/* Status of Disk Functions */
typedef BYTE DSTATUS;

/* Results of Disk Functions */
typedef enum
{
    RES_OK = 0, /* 0: Function succeeded */
    RES_ERROR,  /* 1: Disk error */
    RES_NOTRDY, /* 2: Not ready */
    RES_PARERR  /* 3: Invalid parameter */
} DRESULT;

class DISKIO
{
  private:
    /**
     * @brief Enable SD chip for SPI communication. Set SD_CS pin low.
     */
    static void enable();

    /**
     * @brief Disable SD chip for SPI communication. Set SD_CS pin
        high.
     */
    static void disable();

    /**
     * @brief Send a command packet to MMC
     *
     * @param cmd 1st byte (Start + Index)
     * @param arg Argument (32 bits)
     * @return BYTE response value
     */
    static BYTE send_cmd(BYTE cmd, DWORD arg);

  public:
    static BYTE CardType;

    /**
     * @brief Construct a new DISKIO object
     *
     */
    DISKIO();

    /**
     * @brief Initialize Disk Drive
     *
     * @return DSTATUS Initialize result
     */
    static DSTATUS disk_initialize();

    /**
     * @brief Read partial sector
```

```
69          *
70          * @param buff Pointer to the read buffer
71          * @param lba Sector number (LBA)
72          * @param ofs Byte offset to read from (0..511)
73          * @param cnt Number of bytes to read (ofs + cnt mus be <= 512)
74          * @return DRESULT read result
75          */
76         static DRESULT disk_readp(BYTE*, DWORD, WORD, WORD);
77
78         /**
79          * @brief Write partial sector
80          *
81          * @param buff Pointer to the bytes to be written
82          * @param sa Number of bytes to send, Sector number (LBA) or zero
83          * @return DRESULT write result
84          */
85         static DRESULT disk_writep(const BYTE*, DWORD);
86 };
87
88 #define _DISKIO
89 #endif
```

Listing 2: DISKIO Class.

4.3.3 INTERRUPTIONS

```
1  /**
2   * @file interruptions.hh
3   * @author Jakub Kozłowicz, Aleksandra Rozmus
4   * @brief Header file for digital measurement module
5   * @version 0.1
6   * @date 2022-01-25
7   *
8   * @copyright Copyright (c) 2021
9   *
10  */
11
12 #include <avr/interrupt.h>
13 #include <avr/io.h>
14 #include <util/delay.h>
15
16 #ifndef INTERRUPTIONS_HH_
17 #define INTERRUPTIONS_HH_
18
19 #if(!defined _SW_OK || !defined _SW_NOK)
20 #define _SW_OK  PD2
21 #define _SW_NOK PD3
22 #endif
23
24 class INTERRUPT
25 {
26   private:
27     /**
28      * @brief
29      *
30      */
31     static uint8_t ok;
32
33     /**
34      * @brief
35      *
36      */
37     static uint8_t nok;
```

```
38
39    public:
40      /**
41       * @brief Construct a new INTERRUPT object
42       *
43       */
44      INTERRUPT();
45
46      /**
47       * @brief check status of ok
48       *
49       * @return true if ok is 1
50       * @return false if ok is 0
51       */
52      bool is_ok() { return ok == 1; };
53
54      /**
55       * @brief check status ognok
56       *
57       * @return true if nok is 1
58       * @return false if ok is 0
59       */
60      bool is_nok() { return nok == 1; };
61
62      /**
63       * @brief Set ok to 0
64       *
65       */
66      void clear_ok() { ok = 0; };
67
68      /**
69       * @brief Set nok to 0
70       *
71       */
72      void clear_nok() { nok = 0; };
73
74      /**
75       * @brief Set ok to 1
76       *
77       */
78      static void set_ok() { ok = 1; };
79
80      /**
81       * @brief Set nok to 1
82       *
83       */
84      static void set_nok() { nok = 1; };
85
86      /**
87       * @brief Destroy the INTERRUPT object
88       *
89       */
90      ~INTERRUPT(){};
91  };
92
93  #endif /* INTERRUPTIONS_HH_ */
```

Listing 3: INTERRUPTIONS Class.

4.3.4 LCD

```
1  /**
2   * @file lcd.hh
```

```
 3  * @author Jakub Kozłowicz, Aleksandra Rozmus
 4  * @brief Header file for LCD module
 5  * @version 0.1
 6  * @date 2021-10-21
 7  *
 8  * @copyright Copyright (c) 2021
 9  *
10  */
11
12 #include <avr/io.h>
13 #include <util/delay.h>
14
15 #ifndef LCD_HH_
16 #define LCD_HH_
17
18 #if(!defined _LCD_DDR || !defined _LCD_PORT)
19 #define _LCD_DDR  DDRC
20 #define _LCD_PORT PORTC
21 #endif
22
23 #if(!defined _LCD_RS || !defined _LCD_EN)
24 #define _LCD_RS 0
25 #define _LCD_RW 1
26 #define _LCD_EN 2
27 #endif
28
29 #if(!defined _LCD_D4 || !defined _LCD_D5 || !defined _LCD_D6 ||
       !defined _LCD_D7)
30 #define _LCD_D4 4
31 #define _LCD_D5 5
32 #define _LCD_D6 6
33 #define _LCD_D7 7
34 #endif
35
36 #define _LCD_COL_COUNT 16
37 #define _LCD_ROW_COUNT 2
38
39 #define _LCD_CLEAR_DISPLAY 0x01
40 #define _LCD_4BIT_MODE     0x02
41 #define _LCD_8BIT_MODE     0x10
42 #define _LCD_2LINE_4BIT    0x28
43 #define _LCD_2LINE_8BIT    0x38
44 #define _LCD_CURSOR_OFF    0x0C
45 #define _LCD_INCREMENT     0x06
46 #define _LCD_FIRST_LINE    0x80
47 #define _LCD_SECOND_LINE   0xC0
48 #define _LCD_SHIFT_RIGHT   0x1C
49 #define _LCD_SHIFT_LEFT    0x18
50
51 class LCD
52 {
53   private:
54     /**
55      * @brief Send the command value to the LCD data port
56      * @param cmd command value
57      */
58     void command(uint8_t cmd);
59
60     /**
61      * @brief Send command to the data port
62      * @param sign Individual character from string
63      */
64     void write(uint8_t sign);
65
66   public:
```

```
67      /**
68       * @brief Construct a new LCD object
69       */
70      LCD();
71
72      /**
73       * @brief Clear display and set cursor at home position
74       */
75      void clear();
76
77      /**
78       * @brief Shows the text on the display
79       * @param string String with text
80       */
81      void display(uint8_t string[]);
82
83      /**
84       * @brief Set the position of text
85       * @param y Number of row
86       * @param x Number of column
87       */
88      void set_position(uint8_t y, uint8_t x);
89
90      /**
91       * @brief Moves text to the left
92       */
93      void shift_left();
94
95      /**
96       * @brief Moves text to the right
97       */
98      void shift_right();
99
100     /**
101      * @brief Move text forward and backward
102      */
103     void roll();
104
105     /**
106      * @brief Display text on selected position
107      * @param y Number of row
108      * @param x Number of column
109      * @param string String with text
110      */
111     void display_on_pos(uint8_t y, uint8_t x, uint8_t string[]);
112
113     /**
114      * @brief Display text in hex on position
115      * @param y Number of row
116      * @param x Number of column
117      * @param d Hex byte to display
118      */
119     void display_on_pos_hex(uint8_t y, uint8_t x, uint8_t d);
120
121     /**
122      * @brief Destroy the LCD object
123      */
124     ~LCD(){};
125 };
126
127 #endif /* LCD_HH_ */
```

Listing 4: LCD Class.

4.3.5 LED

```
/**
 * @file led.hh
 * @author Jakub Kozłowicz, Aleksandra Rozmus
 * @brief Header file for LED module
 * @version 0.1
 * @date 2021-10-21
 *
 * @copyright Copyright (c) 2021
 *
 */

#include <avr/io.h>
#include <util/delay.h>

#ifndef LED_HH_
#define LED_HH_

#if(!defined _LED_DDR || !defined _LED_PORT)
#define _LED_DDR DDRA
#define _LED_PORT PORTA
#endif

#if(!defined _LED_RED || !defined _LED_GREEN || !defined _LED_YELLOW)
#define _LED_RED 5
#define _LED_YELLOW 6
#define _LED_GREEN 7
#endif

class LED
{
  private:
  public:
    /**
     * @brief Construct a new LED object
     */
    LED();

    /**
     * @brief Turn on specific LED
     * @param led_pin LEDs pin number
     */
    void turn_on(int led_pin);

    /**
     * @brief Turn off specific LED
     * @param led_pin LEDs pin number
     */
    void turn_off(int led_pin);

    /**
     * @brief Destroy the LED object
     */
    ~LED(){};
};

#endif /* LED_HH_ */
```

Listing 5: LED Class.

4.3.6 PFF

```
/*-----------------------------------------------------------/
/  Petit FatFs - FAT file system module include file  R0.02a (C)ChaN, 2010
/-----------------------------------------------------------/
/ Petit FatFs module is an open source software to implement FAT file system to
/ small embedded systems. This is a free software and is opened for education,
/ research and commercial developments under license policy of following trems.
/
/  Copyright (C) 2010, ChaN, all right reserved.
/
/ * The Petit FatFs module is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/   personal, non-profit or commercial use UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
/-----------------------------------------------------------*/

/**
 * @file pff.hh
 * @author Jakub Kozłowicz, Aleksandra Rozmus
 * @brief Header file for PFF module
 * @version 0.1
 * @date 2022-01-10
 *
 * @copyright Copyright (C) 2010, ChaN
 *
 */

#include "diskio.hh"

/*-----------------------------------------------------/
/ Petit FatFs Configuration Options
/
/ CAUTION! Do not forget to make clean the project after any changes to
/ the configuration options.
/
/-----------------------------------------------------*/
#ifndef _FATFS
#define _FATFS

#define _USE_READ   1 /* 1:Enable read() */
#define _USE_DIR    0 /* 1:Enable opendir() and readdir() */
#define _USE_LSEEK  0 /* 1:Enable lseek() */
#define _USE_WRITE  0 /* 1:Enable write() */
#define _FS_FAT12   0 /* 1:Enable FAT12 support */
#define _FS_FAT32   1 /* 1:Enable FAT32 support */
#define _CODE_PAGE 1
/* Defines which code page is used for path name. Supported code pages are:
/  932, 936, 949, 950, 437, 720, 737, 775, 850, 852, 855, 857, 858, 862, 866,
/  874, 1250, 1251, 1252, 1253, 1254, 1255, 1257, 1258 and 1 (ASCII only).
/  SBCS code pages except for 1 requiers a case conversion table. This
/  might occupy 128 bytes on the RAM on some platforms, e.g. avr-gcc. */

#define _WORD_ACCESS 0
/* The _WORD_ACCESS option defines which access method is used to the word
/  data in the FAT structure.
```

```
56 /
57 /   0: Byte-by-byte access. Always compatible with all platforms.
58 /   1: Word access. Do not choose this unless following condition is
       met.
59 /
60 /  When the byte order on the memory is big-endian or address
       miss-aligned
61 /  word access results incorrect behavior, the _WORD_ACCESS must be
       set to 0.
62 /  If it is not the case, the value can also be set to 1 to improve the
63 /  performance and code efficiency. */
64
65 /* End of configuration options. Do not change followings without
       care.     */
66 /*---------------------------------------------*/
67
68 #if _FS_FAT32
69 #define CLUST DWORD
70 #else
71 #define CLUST WORD
72 #endif
73
74 /* File system object structure */
75
76 typedef struct
77 {
78     BYTE fs_type; /* FAT sub type */
79     BYTE flag;    /* File status flags */
80     BYTE csize;   /* Number of sectors per cluster */
81     BYTE pad1;
82     WORD n_rootdir;   /* Number of root directory entries (0 on FAT32)
          */
83     CLUST n_fatent;   /* Number of FAT entries (= number of clusters +
          2) */
84     DWORD fatbase;    /* FAT start sector */
85     DWORD dirbase;    /* Root directory start sector (Cluster# on
          FAT32) */
86     DWORD database;   /* Data start sector */
87     DWORD fptr;       /* File R/W pointer */
88     DWORD fsize;      /* File size */
89     CLUST org_clust;  /* File start cluster */
90     CLUST curr_clust; /* File current cluster */
91     DWORD dsect;      /* File current data sector */
92 } FATFS;
93
94 /* Directory object structure */
95
96 typedef struct
97 {
98     WORD index;   /* Current read/write index number */
99     BYTE* fn;     /* Pointer to the SFN (in/out)
          {file[8],ext[3],status[1]} */
100    CLUST sclust; /* Table start cluster (0:Static table) */
101    CLUST clust;  /* Current cluster */
102    DWORD sect;   /* Current sector */
103 } DIR;
104
105 /* File status structure */
106
107 typedef struct
108 {
109     DWORD fsize;    /* File size */
110     WORD fdate;     /* Last modified date */
111     WORD ftime;     /* Last modified time */
112     BYTE fattrib;   /* Attribute */
```

```
113     char fname[13]; /* File name */
114 } FILINFO;
115
116 /* File function return code (FRESULT) */
117
118 typedef enum
119 {
120     FR_OK = 0,          /* 0 */
121     FR_DISK_ERR,        /* 1 */
122     FR_NOT_READY,       /* 2 */
123     FR_NO_FILE,         /* 3 */
124     FR_NO_PATH,         /* 4 */
125     FR_NOT_OPENED,      /* 5 */
126     FR_NOT_ENABLED,     /* 6 */
127     FR_NO_FILESYSTEM    /* 7 */
128 } FRESULT;
129
130 class PFF
131 {
132   private:
133     /**
134      * @brief Pointer to the file system object (logical drive)
135      *
136      */
137     static FATFS* FatFs;
138
139     /**
140      * @brief Fill memory
141      *
142      * @param dst pointer to the destination object
143      * @param val value
144      * @param cnt number of bytes to copy
145      */
146     static void mem_set(void* dst, int val, int cnt);
147
148     /**
149      * @brief Compare memory to memory
150      *
151      * @param dst pointer to the destination object
152      * @param src pointer to the source object
153      * @param cnt number of bytes to copy
154      * @return int the pointer to the destination buffer
155      */
156     static int mem_cmp(const void* dst, const void* src, int cnt);
157
158     /**
159      * @brief FAT access - Read value of a FAT entry
160      *
161      * @param clst Cluster# to get the link information
162      * @return CLUST 1:IO error, Else:Cluster status
163      */
164
165     static CLUST get_fat(CLUST clst);
166     /**
167      * @brief Get sector# from cluster#
168      *
169      * @param clst Cluster# to be converted
170      * @return DWORD !=0: Sector number, 0: Failed - invalid cluster#
171      */
172     static DWORD clust2sect(CLUST clst);
173
174     /**
175      * @brief Directory handling - Rewind directory index
176      *
177      * @param dj Pointer to directory object
```

```
178         * @return FRESULT Seek result
179         */
180        static FRESULT dir_rewind(DIR* dj);
181
182        /**
183         * @brief Directory handling - Move directory index next
184         *
185         * @param dj Pointer to directory object
186         * @return FRESULT FR_OK:Succeeded, FR_NO_FILE:End of table
187         */
188        static FRESULT dir_next(DIR* dj);
189
190        /**
191         * @brief Directory handling - Find an object in the directory
192         *
193         * @param dj Pointer to the directory object linked to the file
                name
194         * @param dir 32-byte working buffer
195         * @return FRESULT result
196         */
197        static FRESULT dir_find(DIR* dj, BYTE* dir);
198
199        /**
200         * @brief Pick a segment and create the object name in directory
                form
201         *
202         * @param dj Pointer to the directory object
203         * @param path Pointer to pointer to the segment in the path string
204         * @return FRESULT result of create the object name
205         */
206        static FRESULT create_name(DIR* dj, const char** path);
207
208        /**
209         * @brief Follow a file path
210         *
211         * @param dj Directory object to return last directory and found
                object
212         * @param dir 32-byte working buffer
213         * @param path Full-path string to find a file or directory
214         * @return FRESULT FR_OK(0): successful, !=0: error code
215         */
216        static FRESULT follow_path(DIR* dj, BYTE* dir, const char* path);
217
218        /**
219         * @brief Check a sector if it is an FAT boot record
220         *
221         * @param buf Working buffer
222         * @param sect Sector# (lba) to check if it is an FAT boot record
                or not
223         * @return BYTE 0:The FAT boot record, 1:Valid boot record but not
                an FAT, 2:Not a boot record, 3:Error
224         */
225        static BYTE check_fs(BYTE* buf, DWORD sect);
226
227    public:
228        /**
229         * @brief Construct a new PFF object
230         *
231         */
232        PFF();
233
234        /**
235         * @brief Mount/Unmount a logical drive
236         *
237         * @param fs Pointer to new file system object description
```

```
238         * @return FRESULT
239         */
240        FRESULT mount(FATFS* fs);
241
242        /**
243         * @brief Open a file
244         *
245         * @param path Pointer to the file name
246         * @return FRESULT open result
247         */
248        FRESULT open(const char* path);
249
250        /**
251         * @brief Read data from the open file
252         *
253         * @param buff Pointer to the read buffer
254         * @param btr Number of bytes to read
255         * @param br Pointer to number of bytes read
256         * @return FRESULT read result
257         */
258        FRESULT read(void* buff, WORD btr, WORD* br);
259
260        /**
261         * @brief Write data to the open file
262         *
263         * @param buff Pointer to the data to be written
264         * @param btw Number of bytes to write (0:Finalize the current
               write operation)
265         * @param bw Pointer to number of bytes written
266         * @return FRESULT write result
267         */
268        FRESULT write(const void* buff, WORD btw, WORD* bw);
269
270        /**
271         * @brief Move file pointer of the open file
272         *
273         * @param ofs File pointer from top of fil
274         * @return FRESULT lseek result
275         */
276        FRESULT lseek(DWORD ofs);
277    };
278
279    #endif /* _FATFS */
```

Listing 6: PFF Class.

4.3.7 PWM

```
1  /**
2   * @file pwm.hh
3   * @author Jakub Kozłowicz, Aleksandra Rozmus
4   * @brief Header file for PWM module
5   * @version 0.1
6   * @date 2021-10-21
7   *
8   * @copyright Copyright (c) 2021
9   *
10  */
11
12  #include <avr/io.h>
13  #include <util/delay.h>
14
15  #ifndef PWM_HH_
```

```
16 #define PWM_HH_
17
18 #if(!defined _PWM_PIN || !defined _PWM_DDR || !defined _PWM_PORT ||
       !defined _PWM_REGISTRY)
19 #define _PWM_PIN       7
20 #define _PWM_DDR       DDRD
21 #define _PWM_PORT      PORTD
22 #define _PWM_REGISTRY  OCR2
23 #endif
24
25 class PWM
26 {
27   private:
28     const uint8_t min_current_duty = 30;
29     const uint8_t max_current_duty = 155;
30     const uint8_t min_voltage_duty = 0;
31     const uint8_t max_voltage_duty = 255;
32
33     void set_duty(int duty);
34
35   public:
36     PWM();
37     void wave();
38     ~PWM(){};
39 };
40
41 #endif /* PWM_HH_ */
```

Listing 7: PWM Class.

4.3.8 SPI

```
1  /**
2   * @file spi.hh
3   * @authors Jakub Kozłowicz, Aleksandra Rozmus
4   * @brief Header file for SPI module
5   * @version 0.1
6   * @date 2021-10-21
7   *
8   * @copyright Copyright (c) 2021
9   *
10  */
11
12 #include <avr/io.h>
13 #include <util/delay.h>
14
15 #ifndef SPI_HH_
16 #define SPI_HH_
17
18 #define _SPI_DDR  DDRB
19 #define _SPI_PORT PORTB
20
21 #define _SPI_SS   4
22 #define _SPI_MOSI 5
23 #define _SPI_MISO 6
24 #define _SPI_SCK  7
25
26 #if(!defined _SPI_CS_1 || !defined _SPI_CS_2)
27 #define _SPI_CS_1 0
28 #define _SPI_CS_2 1
29 #endif
30
31 class SPI
```

```
32 {
33   public:
34     /**
35      * @brief Construct a new SPI object
36      * @details Set directions on all pins and initialize them with
           proper
37      * value. Enable SPI communication in master mode with speed of
           Fosc/2
38      */
39     SPI();
40
41     /**
42      * @brief Send data to slave device
43      * @param data Data to be transmitted
44      * @return uint8_t - Value of SPDR register
45      */
46     static uint8_t master_transmit(uint8_t data);
47
48     /**
49      * @brief Read data from slave device
50      * @return uint8_t - Value of SPDR register
51      */
52     static uint8_t master_read();
53
54     /**
55      * @brief Destroy the SPI object
56      * @details Dummy descructor
57      */
58     ~SPI(){};
59 };
60
61 #endif /* SPI_HH_ */
```

Listing 8: SPI Class.

4.3.9 RFID

```
1  /**
2   * @file rfid.hh
3   * @authors Jakub Kozłowicz, Aleksandra Rozmus
4   * @brief Header file for RFID scanner module
5   * @version 0.1
6   * @date 2021-10-21
7   *
8   * @Copyright Copyright (C) 2021
9   *
10  */
11
12 #include <avr/io.h>
13 #include <util/delay.h>
14
15 #include "rfid_reg.hh"
16 #include "spi.hh"
17
18 #ifndef RFID_HH_
19 #define RFID_HH_
20
21 #if(!defined _RFID_DDR || !defined _RFID_PORT)
22 #define _RFID_DDR  DDRB
23 #define _RFID_PORT PORTB
24 #define _RFID_PIN  PINB
25 #endif
26
```

```
27 #if(!defined _CS_RFID || !defined _RFID_RST)
28 #define _RFID_SS  _SPI_CS_2
29 #define _RFID_RST 2
30 #endif
31
32 /**
33  * @brief Structure used for passing the UID of a PICC.
34  */
35 struct UID
36 {
37     uint8_t size;         /** @brief Number of bytes in the UID; 4, 7,
        10 */
38     uint8_t uid_byte[10]; /** @brief Array to store uid bytes. */
39     uint8_t sak;          /** @brief The SAK (Select acknowledge) byte
        returned from PICC successful selection. */
40 };
41
42 class RFID
43 {
44   private:
45     /**
46      * @brief Structure to save informations about card uid
47      */
48     UID uid;
49
50     /**
51      * @brief Enable RFID chip for SPI communication
52      */
53     void enable();
54
55     /**
56      * @brief Disable RFID chip for SPI communication
57      */
58     void disable();
59
60     /**
61      * @brief Turns the antenna on by enabling pins TX1 and TX2.
62      */
63     void antenna_on();
64
65     /**
66      * @brief Sets the bits given in mask in register reg.
67      * @param reg the register to update
68      * @param mask the bits to set
69      */
70     void set_bit_mask(uint8_t reg, uint8_t mask);
71
72     /**
73      * @brief Clears the bits given in mask from register reg.
74      * @param reg the register to update
75      * @param mask the bits to clear
76      */
77     void clear_bit_mask(uint8_t reg, uint8_t mask);
78
79     /**
80      * @brief Reads a byte from the specified register in the MFRC522
         chip.
81      * @param address the register to read from
82      * @return uint8_t byte read from register
83      */
84     uint8_t MFRC522_read(uint8_t address);
85
86     /**
87      * @brief Reads a number of bytes from the specified register in
         the MFRC522 chip.
```

```
88      * @details The interface is described in the datasheet section
            8.1.2.
89      * @param address the register to read from
90      * @param count the number of bytes to read
91      * @param values byte array to store the values in
92      * @param rx_align only bit positions rxAlign..7 in values[0] are
            updated
93      */
94     void MFRC522_read(uint8_t address, uint8_t count, uint8_t* values,
           uint8_t rx_align);
95
96     /**
97      * @brief Writes a byte to the specified register in the MFRC522
            chip.
98      * @param address the register to write to
99      * @param value the value to write
100     */
101    void MFRC522_write(uint8_t address, uint8_t value);
102
103    /**
104     * @brief Writes a n bytes to the specified register in the
            MFRC522 chip.
105     * @param address the register to write to
106     * @param count number of bytes to write
107     * @param values the values to write
108     */
109    void MFRC522_write(uint8_t address, const uint8_t count, uint8_t*
           const values);
110
111    /**
112     * @brief Communicate with the card.
113     * @details Transfers data to the MFRC522 FIFO, executes a
            command, waits for completion and transfers data back
114     * from the FIFO. CRC validation can only be done if backData and
            backLen are specified.
115     * @param command the command to execute
116     * @param waitIRq the bits in the ComIrqReg register that signals
            succesful completion of the command
117     * @param send_data pointer to the data to transfer to the FIFO
118     * @param send_length number of bytes to transfer to the FIFO
119     * @param back_data nullptr or pointer to the buffer if data
            should be read back after executing the command
120     * [default value = nullptr]
121     * @param back_length IN: max number of bytes to write to
            *back_data; OUT: the number of bytes returned [default
122     * value = nullptr]
123     * @param valid_bits IN/OUT: the number of valid bits in the last
            byte; 0 for 8 valid bits [default value = nullptr]
124     * @param rx_align defines the bit position in back_data[0] for
            the first bit received [default value = 0]
125     * @param check_CRC TRUE => the last two bytes of the response is
            assumed to be a CRC_A that mast be validated
126     * [default value = false]
127     */
128    uint8_t MFRC522_communicate(uint8_t command, uint8_t waitIRq,
           uint8_t* send_data, uint8_t send_length,
129                                uint8_t* back_data = nullptr, uint8_t*
                                   back_length = nullptr,
130                                uint8_t* valid_bits = nullptr, uint8_t
                                   rx_align = 0, bool check_CRC =
                                   false);
131
132    /**
133     * @brief Executes the Transceive command.
```

```
134      * @details CRC validation can only be done if back_data and
             back_length are specified.
135      * @param send_data pointer to the data to transfer to the FIFO
136      * @param send_length number of bytes to transfer to the FIFO
137      * @param back_data nullptr or pointer to the buffer if data
             should be read back after executing the command
138      * [default value = nullptr]
139      * @param back_length IN: max number of bytes to write to
             *back_data; OUT: the number of bytes returned [default
140      * value = nullptr]
141      * @param valid_bits IN/OUT: the number of valid bits in the last
             byte; 0 for 8 valid bits [default value = nullptr]
142      * @param rx_align defines the bit position in back_data[0] for
             the first bit received [default value = 0]
143      * @param check_CRC TRUE => the last two bytes of the response is
             assumed to be a CRC_A that mast be validated
144      * [default value = false]
145      */
146     uint8_t MFRC522_transceive_data(uint8_t* send_data, uint8_t
            send_length, uint8_t* back_data = nullptr,
147                                     uint8_t* back_length = nullptr,
                                           uint8_t* valid_bits = nullptr,
                                           uint8_t rx_align = 0,
148                                     bool check_CRC = false);
149
150     /**
151      * @brief Transmits REQA or WUPA commands.
152      * @param command the command to send
153      * @param buffer_answer_request the buffer to store ATQA (Answer
             to request) in
154      * @param buffer_size buffer size; at least two bytes; also number
             of bytes returned if success
155      * @return uint8_t - Status of request
156      */
157     uint8_t MFRC522_request_wakeup(uint8_t command, uint8_t*
            buffer_answer_request, uint8_t* buffer_size);
158
159     /**
160      * @brief Transmit a REQuest command, type A.
161      * @details Invites PICCs in state IDLE to go to READY and prepare
             for anticollision or selection. 7 bit frame.
162      * @param buffer_answer_request the buffer to store ATQA (Answer
             to request) in
163      * @param buffer_size buffer size; at least two bytes; also number
             of bytes returned if success
164      * @return uint8_t - Status of request
165      */
166     uint8_t request(uint8_t* buffer_answer_request, uint8_t*
            buffer_size);
167
168     /**
169      * @brief Transmits a WakeUP command, type A.
170      * @details Invites PICCs in state IDLE and HALT to go to READY(*)
             and prepare for anticollision or selection. 7 bit
171      * frame.
172      * @param buffer_answer_request the buffer to store ATQA (Answer
             to request) in
173      * @param buffer_size buffer size; at least two bytes; also number
             of bytes returned if success
174      * @return uint8_t - Status of request
175      */
176     uint8_t wake_up(uint8_t* buffer_answer_request, uint8_t*
            buffer_size);
177
178     /**
```

```
179       * @brief Use the CRC coprocessor in the MFRC522 to calculate a
              CRC_A.
180       * @param data pointer to the data to transfer to the FIFO for CRC
              calculation
181       * @param length the number of bytes to transfer
182       * @param result pointer to result buffer; result is written to
              result[0..1], low byte first
183       * @return uint8_t - Status of calculations
184       */
185      uint8_t calculate_CRC(uint8_t* data, uint8_t length, uint8_t*
            result);
186
187      /**
188       * @brief Performs a soft reset on the MFRC522 chip and waits for
              it to be ready again.
189       */
190      void reset();
191
192      /**
193       * @brief Transmits SELECT/ANTICOLLISION commands to select a
              single PICC.
194       * @details Before calling this function the PICCs must be placed
              in the READY(*) state by calling request() or
195       * wake_up(). On success:
196       *      - The chosen PICC is in state ACTIVE(*) and all other PICCs
              have returned to state IDLE/HALT. (Figure 7 of
197       * the ISO/IEC 14443-3 draft.)
198       *      - The UID size and value of the chosen PICC is returned in
              *uid along with the SAK.
199       * @param uid structure containing uid of the card, size of the
              uid, SAK
200       * @param valid_bits the number of known UID bits supplied in
              *uid. If set you must also supply uid->size. [default
201       * value = 0]
202       */
203      uint8_t select_picc(UID* uid, uint8_t valid_bits = 0);
204
205      /**
206       * @brief Instructs a PICC in state ACTIVE(*) to go to state HALT.
207       * @return uint8_t - Status of this action (_RFID_STATUS_OK on
              succes)
208       */
209      uint8_t halt();
210
211    public:
212      /**
213       * @brief Construct a new RFID object
214       * @details In case of setting RST pin perform hard reset or soft
              reset. Set proper speed values and set up
215       * registers. At the end enable antena after either reset.
216       */
217      RFID();
218
219      /**
220       * @brief Check if card is present.
221       */
222      bool is_card();
223
224      /**
225       * @brief Read UID of the card.
226       * @details Read the UID of placed card and return status of this
              reading. Only true on sucess.
227       */
228      bool read_card_serial();
229
```

```
230     /**
231      * @brief Return size of the card UID
232      */
233     uint8_t get_uid_size();
234 
235     /**
236      * @brief Return array with UID.
237      * @details Return UID array converted to decinals. Call halt()
            automaticly.
238      */
239     uint8_t* get_card_uid();
240 
241     /**
242      * @brief Return a version of RFID card reader
243      * @return uint8_t version byte
244      */
245     uint8_t version();
246 
247     /**
248      * @brief Destroy the RFID object
249      * @details Dummy desctructor
250      */
251     ~RFID(){};
252 };
253 
254 #endif /* RFID_HH_ */
```

Listing 9: RFID Class.

Spis rysunków

Spis tablic

Listings