31.01.2021

Projekt sterownika do matrycy LED 7 x 21

Opiekun projektu: dr hab. inż. Zbigniew Świder
Wykonał: Bazan Michał

1. Krótki opis projektu.

Celem podjęcia się tego projektu było utrwalenie dotychczas zdobytej wiedzy i wykorzystanie jej do rozwiązania praktycznego problemu. Dziedziny nauki, których zgłębianie było niezbędne do wykonania automatu to:

- sterowniki mikroprocesorowe i systemy wbudowane,
- automatyka,
- elektronika.

Opisywany sterownik to oprogramowanie mikroprocesora oraz układ elektroniczny składający się z:

- części analogowej odpowiedzialnej za odpowiednie wysterowanie napięcia zasilania matrycy LED,
- części cyfrowej, która ma na celu multipleksowanie sygnału pochodzącego z części analogowej oraz jest odpowiedzialna za relatywnie dokładne utrzymywanie godziny i daty wyświetlanej na matrycy,
- układu mikroprocesorowego, w którego skład wchodzą najprostsze komponenty, takie jak kondensatory, rezystory oraz mikroprocesor.

Opis oprogramowania znajduje się w innej sekcji.

Sterownik w połączeniu z matrycą LED, pozwala na wykorzystanie układu do celów praktycznych jak wskazywanie aktualnej godziny, daty oraz jako budzik, który udostępnia pięć budzików pięć budzików, które po wyłączeniu alarmu pozostają nieaktywne do następnych nastaw, oprogramowanie pozwala na ustawienie długości drzemki w zakresie od jednej do sześćdziesięciu minut.
Poza użytkiem praktycznym, matryca wraz ze sterownikiem stanowi ciekawą ozdobę.

Obsługa tego urządzenia jest bardzo prosta i intuicyjna. Instrukcja postępowania znajduje się w następnym rozdziale.

2. Instrukcja obsługi.

Aby wejść do menu nastaw należy nacisnąć przycisk „FUNCTION” i przytrzymać go przez co najmniej dwie sekundy. Podczas oczekiwania na przejście do menu nastaw na matrycy wyświetla się napis „HOLD”, który sugeruje, aby przycisk był wciśnięty. Po upływie dwóch sekund użytkownik znajduje się w menu nastaw. Oczom użytkownika powinna się ukazać pierwsza możliwa nastawa tj. „TIME”, czyli ustawienia związane ze zliczanym przez układ czasem.

Po przejściu do menu, użytkownik może nawigować przyciskami „+” oraz „-”. Przyciśnięcie któregokolwiek z przycisków nawigacji zmienia wyświetlaną wartość wyłącznie o jedną opcje. Przewijanie listy nastaw podczas gdy wciśnięty jest przycisk jest zablokowane.

Wybranie interesującej użytkownika nastawy odbywa się poprzez wciśnięcie przycisku „OK”.

UWAGA: wielkości takie jak godzina, data czy czas drzemki są zapisywane natychmiast po każdej zmianie przez przyciski nawigacyjne. Po braku aktywności przez piętnaście sekund następuje wyjście do menu głównego a po dwudziestu sekundach braku aktywności następuje wyjście z menu. Podczas, gdy użytkownik jest w menu nastaw budzików, czy brzęczyka, po upłynięciu piętnastu sekund od ostatniej aktywności **ZMIANY NIE SĄ ZAPISYWANE**.

2.1. Ustawianie czasu – „TIME”.

Po wybraniu opcji „TIME”, możliwe jest ustawienie bieżącej godziny, liczby minut oraz wyjście do głównego menu.

Wielkość, która będzie ustawiana po zatwierdzeniem przyciskiem „OK” jest wyświetlana. Po wybraniu interesującej użytkownika opcji i wciśnięciu przycisku „OK” wyświetlane są:

- XX : HH,
- MM : XX, gdzie XX to aktualnie ustawiana wielkość, HH – sugestia, że ustawiana jest liczba godzin, MM sugestia, że nastawiana jest liczba minut.

Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji a zatwierdzenie przyciskiem „OK” pozwala powrócić użytkownikowi do podmenu.

Wybranie opcji „EXIT” pozwala wyjść do menu głównego nastaw.

2.2. Ustawianie aktualnej daty – „DATE”.

Po wybraniu opcji „DATE”, możliwe jest ustawienie bieżącej daty, oraz wyjście do głównego menu.

Wielkość, która będzie ustawiana po zatwierdzeniem przyciskiem „OK” jest wyświetlana. Po wybraniu interesującej użytkownika opcji i wciśnięciu przycisku „OK” wyświetlane są:

- XX : DD,
- MM : XX,
- 20XX, gdzie XX to aktualnie ustawiana wielkość, DD to sugestia, że nastawiany jest dzień miesiąca, MM to sugestia, że aktualnie ustawiany jest miesiąc.

20XX – możliwość nastawy roku znajduje się w przedziale od 2000 do 2099, co jest spowodowane ograniczeniami układu czasu rzeczywistego **DS1307**.

Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji a zatwierdzenie przyciskiem „OK” pozwala powrócić użytkownikowi do podmenu.

Wybranie opcji „EXIT” pozwala wyjść do menu głównego nastaw.

2.3. Ustawianie długości drzemki – „BUZZ”.

Możliwość włączenia brzęczyka, który powiadamia brzęczeniem o rozpoczętej nowej godzinie. Możliwe nastawy:

- „OFF” – wyłączony,
- „ON” – włączony,
- „EXIT” – wyjście z podmenu.

2.4. Ustawianie długości drzemki – „SNZE”.

Po wybraniu opcji „SNZE”, możliwe jest ustawienie długości drzemki alarmu (opis postępowania z alarmem w innym podrozdziale).

Po wejściu do tego podmenu dostępne są następujące opcje:

- XXMM,
- „EXIT”, gdzie XX to puste, białe znaki (NULL), MM to ilość minut określająca długość drzemki.

Po zatwierdzeniu „XXMM” przyciskiem „OK”, wyświetlana jest następująca kombinacja „SNMM”, gdzie SN ma sugerować, że ustawiana wielkość to drzemka.

Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji a zatwierdzenie przyciskiem „OK” pozwala powrócić użytkownikowi do podmenu.

Wybranie opcji „EXIT” pozwala wyjść do menu głównego nastaw.

2.5. Ustawianie jednorazowego alarmu – „S_AL”.

Po wybraniu tej opcji, wyświetla się menu, po którym użytkownik może nawigować przyciskami „+” i „-” aby wybrać pożądany alarm lub wyjść z podmenu. Po wybraniu dowolnego alarmu dostępne są następujące opcje:

- „OFF” – wyłącza alarm,
- „ON” – włącza alarm,
- „SET” – przejście do ustawienia godziny,
- „EXIT” – wyjście z nastaw danego alarmu.

UWAGA: gdy alarm jest aktywny, wciśnięcie przycisku „+” wywołuje drzemkę a „OK” wyłącza alarm. Po dokonaniu nastaw danego alarmu jest on automatycznie aktywowany.

2.6. Wyjście z menu nastaw – „EXIT”.

Zatwierdzenie tej opcji powoduje wyjście z menu nastaw i normalny stan pracy urządzenia w sekwencji wyświetlanie godziny (dwadzieścia sekund), daty (pięć sekund, format dd-mm) oraz roku (przez pięć sekund).

3. Dokumentacja techniczna.

Niniejsza sekcja zaczyna się od opisu oprogramowania a kończy na układzie elektronicznym. Nie wszystkie pliki są opisane w całości. Aby dokonać głębszej analizy, należy udać się na stronę, na której znajduje się repozytorium:

https://github.com/DevxMike/led_matrix

3.1. Opis operacji niskopoziomowych:

- Komunikacja I2C

```
#include <avr/io.h>
#include "data_structs.h"
#ifndef twi_h
#define twi_h
#define SLAVE 0xD0
uint8_t write_buff(uint8_t, uint8_t, uint8_t, const uint8_t*);
void read_buff(uint8_t, uint8_t, uint8_t, uint8_t*);

uint8_t init_RTC(void);
inline uint8_t bcd_to_dec(uint8_t bcd){
return (bcd & 0x0F) + 10 * ((bcd >> 4) & 0x0F);
}
inline uint8_t dec_to_bcd(uint8_t dec){
return ((dec/10) << 4) | (dec % 10);
}
uint8_t day_of_week(uint16_t y, uint8_t m, uint8_t d);
inline uint8_t is_leap(uint16_t year) {
return (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0));
}
#endif
```

Plik „twi.h”.

Plik zawiera definicje funkcji konwertujących kod bcd na kod dziesiętny, kod dziesiętny na kod bcd oraz funkcję do sprawdzenia, czy rok jest przestępny. W pliku znajdują się również deklaracje funkcji realizujących inicjalizację układu RTC, przesłanie bufora informacji, odczytania bufora informacji z układu czasu rzeczywistego oraz funkcji określającej dzień tygodnia.

```
#include "twi.h"

void TWI_START(void){
TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTA); //enable TWI, clear TWINT and START
while(!(TWCR & (1 << TWINT)));
}
void TWI_STOP(void){
TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO); //clear TWINT, enable TWI and STOP
return;
while(!(TWCR & (1 << TWINT)));
}
void TWI_WRITE(uint8_t byte){
TWDR = byte; //place byte desired to send into reg
TWCR = (1 << TWINT) | (1 << TWEN); //start transmission
while(!(TWCR & (1 << TWINT)));
}
uint8_t TWI_READ(uint8_t ack_bit){
TWCR = (1 << TWINT) | (1 << TWEN) | (ack_bit << TWEA);
while(!(TWCR & (1 << TWINT)));
return TWDR;
}
uint8_t write_buff(uint8_t slave, uint8_t ram_adr, uint8_t len, const uint8_t* buff){
TWI_START();
TWI_WRITE(slave);
TWI_WRITE(ram_adr);
while(len--){
TWI_WRITE(*buff++);
}
TWI_STOP();
return 0;
}
void read_buff(uint8_t slave, uint8_t ram_adr, uint8_t len, uint8_t* buff){
TWI_START();
TWI_WRITE(slave);
TWI_WRITE(ram_adr);
TWI_START();
TWI_WRITE(slave + 1);
while(len--){
*buff++ = TWI_READ(len != 0? 1 : 0);
}
TWI_STOP();
}
uint8_t init_RTC(void){
uint8_t buff[8] = { 0x00, 0x59, 0x00, 0x01, 0x25, 0x01, 0x21, 0x00 };
write_buff(SLAVE, 0x00, 8, buff);
return 0;
}
uint8_t day_of_week(uint16_t y, uint8_t m, uint8_t d){
```

```
static int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
if( m < 3 ) {
y -= 1;
}
return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;
}
```

Plik „twi.c”.

Plik zawiera definicje funkcji, których deklaracje znajdują się w pliku „twi.h” oraz tych, które powinny zostać ukryte przed potencjalnym „użytkownikiem” mojego kodu takie jak np. warunek początku transmisji.

- 16-bit timer

```
#ifndef timers_h
#define timers_h
#include <avr/io.h>
void init_timers(void);
#endif
```

Plik „timers.h”.

Plik zawiera deklaracje funkcji inicjującej 16 bitowy timer sprzętowy.

```
#include "timers.h"
void init_timers(void){
TCCR1A = 0x00;
TCCR1B = (1 << WGM12) | (1 << CS10);
OCR1AH = 0x1F;//2kHz
OCR1AL = 0x3F;
TIMSK |= (1 << OCIE1A);
}
```

Plik „timers.c”

Plik zawiera definicję funkcji inicjalizującej timer sprzętowy. Tryb CTC, preskaler 1, zawartość rejestru przechowującą zawartość wzorcową 0x1F3Fh (7999). Przerwania generowane są z częstotliwością 2kHz.

- SPI

```
#ifndef spi_h
#define spi_h
#include <avr/io.h>
typedef struct{
uint8_t first, second, third;
} reg_data_t;
void init_spi(void);
void send_set(const reg_data_t* data);
#endif
```

Plik „spi.h”.

Plik zawiera definicje typu danych, który przechowuje dane przesyłane do rejestrów matrycy LED oraz deklaracje dwóch funkcji: inicjalizacja SPI oraz przesył trzech bajtów do matrycy.

```
#include "spi.h"
void send_byte(const uint8_t byte){
SPDR = byte;
while(!(SPSR & (1 << SPIF))) { continue; } //wait while transfer is not done
}
inline void set_ss(void){
PORTB |= (1 << PB2);
}
inline void clr_ss(void){
PORTB &= ~(1 << PB2);
}
void send_set(const reg_data_t* data){ //send 3 bytes set
clr_ss();
send_byte(data->third);
send_byte(data->second);
send_byte(data->first);
set_ss();
}
void init_spi(void){
DDRB |= (1 << PB3) | (1 << PB5) | (1 << PB2); //MOSI, SCK and !SS as outputs
SPCR |= (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0) | (1 << DORD); //init spi, master mode, 64 prescaler
}
```

Plik „spi.c”.

Plik zawiera definicje wyżej wymienionych funkcji oraz takich, które nie powinny znaleźć się w globalnej przestrzeni nazw.

- klawisze

```
#ifndef controls_h
#define controls_h
#include <avr/io.h>
void controls_init(volatile uint8_t* dec_port, uint8_t dec, volatile uint8_t* inc_port, uint8_t inc);
void reg_pins(volatile uint8_t* dec_pin, volatile uint8_t* inc_pin);
void function_init(volatile uint8_t* ok_port, uint8_t ok, volatile uint8_t* fn_port, uint8_t fn);
void reg_fn_pins(volatile uint8_t* ok_pin, volatile uint8_t* fn_pin);
extern uint8_t incK, decK, okK, functionK;
void update_controls(void);
#endif
```

Plik „controls.h”.

Plik zawiera deklaracje funkcji potrzebnych do obsługi klawiszy (nie zostały wykorzystane przerwania) oraz zmiennych globalnie dostępnych.

```
#include "controls.h"
volatile uint8_t* _dec_pin, *_inc_pin, * _ok_pin, *_fn_pin; //thats temporary while working on prototype
uint8_t incK = 0, decK = 0, inc_offset, dec_offset;
uint8_t okK = 0, functionK = 0, ok_offset = 0, fn_offset = 0;
void controls_init(volatile uint8_t* dec_port, uint8_t dec, volatile uint8_t* inc_port, uint8_t inc){
*dec_port |= (1 << dec); //pullups
*inc_port |= (1 << inc);
inc_offset = inc;
dec_offset = dec;
}
void reg_pins(volatile uint8_t* dec_pin, volatile uint8_t* inc_pin){
_dec_pin = dec_pin;
_inc_pin = inc_pin;
}
void update_controls(void){
incK = !((*_inc_pin) & (1 << inc_offset));
decK = !((*_dec_pin) & (1 << dec_offset));
okK = !((*_ok_pin) & (1 << ok_offset));
functionK = !((*_fn_pin) & (1 << fn_offset));
}
void function_init(volatile uint8_t* ok_port, uint8_t ok, volatile uint8_t* fn_port, uint8_t fn){
*ok_port |= (1 << ok);
*fn_port |= (1 << fn);
ok_offset = ok;
fn_offset = fn;
}
void reg_fn_pins(volatile uint8_t* ok_pin, volatile uint8_t* fn_pin){
_ok_pin = ok_pin;
_fn_pin = fn_pin;
}
```

Plik „controls.c”.

Plik zawiera definicje funkcji, które pozwalają na podpięcie klawiszy do dowolnego portu procesora. Plik definiuje zmienne wskaźnikowe „ulotne”, których zadaniem jest przechowywanie adresu portu każdego z przycisków, zmienne całkowite, które przechowują wagę bitu danego przycisku.

- zestaw znaków:

```
#ifndef chars_h
#define chars_h
#include <avr/eeprom.h>
#include "spi.h"
enum dots_mode {
none = 0, one, both
};
void prepare_set(uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, reg_data_t*, uint8_t dots);
#endif
```

Plik „chars.h”.

Plik zawiera definicję typu wyliczeniowego, który pomaga określić, ile „kropek” ma zostać wyświetlonych na matrycy (none – zero, one – separator daty, jedna kropka, both – dwie kropki, separator godzin i minut). W pliku zawarta jest również deklaracja funkcji przygotowującej zestaw danych dla matrycy zależnie od tego, co „użytkownik” kodu chce wyświetlić oraz czy potrzebuje kropki.

```
#include "chars.h"
const uint8_t EEMEM characters[][7] = {
{0x1F, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1F}, //0
{0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, //1
{0x1F, 0x01, 0x01, 0x1F, 0x10, 0x10, 0x1F}, //2
{0x1F, 0x01, 0x01, 0x1F, 0x01, 0x01, 0x1F}, //3
{0x11, 0x11, 0x11, 0x1F, 0x01, 0x01, 0x01}, //4
{0x1F, 0x10, 0x10, 0x1F, 0x01, 0x01, 0x1F}, //5
{0x1F, 0x10, 0x10, 0x1F, 0x11, 0x11, 0x1F}, //6
{0x1F, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, //7
{0x1F, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x1F}, //8
{0x1F, 0x11, 0x11, 0x1F, 0x01, 0x01, 0x1F}, //9
{0x0E, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11}, //A, 10
{0x1E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x1E}, //B, 11
{0x1E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1E}, //D, 12
{0x1F, 0x10, 0x10, 0x1F, 0x10, 0x10, 0x1F}, //E, 13
{0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04}, //I, 14
{0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F}, //L, 15
{0x11, 0x1B, 0x15, 0x11, 0x11, 0x11, 0x11}, //M, 16
{0x11, 0x11, 0x19, 0x15, 0x13, 0x11, 0x11}, //N, 17
{0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11}, //R, 18
```

```
{0x0E, 0x10, 0x10, 0x0E, 0x01, 0x01, 0x0E}, //S, 19
{0x1F, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04}, //T, 20
{0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E}, //U, 21
{0x11, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x11}, //X, 22
{0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F}, //Z, 23
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F}, //_, 24
{0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11}, //H, 25
{0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E}, //O, 26
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, //NULL 27
{0x1F, 0x10, 0x10, 0x1F, 0x10, 0x10, 0x10} //F 28
};

void prepare_set(uint8_t first, uint8_t second, uint8_t third, uint8_t fourth, uint8_t row, reg_data_t* set, uint8_t dots){
first = eeprom_read_byte(&characters[first][row]); //read bytes from eemem defined by char codes
second = eeprom_read_byte(&characters[second][row]);
third = eeprom_read_byte(&characters[third][row]);
fourth = eeprom_read_byte(&characters[fourth][row]);

set->first = set->second = set->third = 0x00; //zero out data
set->first |= (first << 3); //set first digit
set->first |= (second & 0xF8) >> 3; //set 2 cols second digit
set->second |= (second & 0x07) << 5; //set second digit
set->second |= (third & 0xFE) >> 1; //4 rows third digit
set->third |= (third & 0x01) << 7; //set third digit
set->third |= fourth << 1;

switch(dots){
case one: if(row == 5) { set->second |= (1 << 4); } break;
case both: if(row == 1 || row == 5) { set->second |= (1 << 4); } break;
break;
}
}
```

Plik „chars.c”.

Plik zawiera definicję tablicy dwuwymiarowej znaków zdefiniowanych dla matrycy przechowywanej w pamięci EEPROM mikroprocesora oraz definicję funkcji przygotowującej zestaw danych dla rejestrów matrycy.

- struktury danych

```
#ifndef data_structs_h
#define data_structs_h
#include <avr/io.h>
#include "controls.h"
#define ALARM_MASK 0x01
enum Month{
january = 1, february, march, april, may,
june, july, august, september, october, november, december
};
typedef struct{
uint8_t hours : 5;
uint8_t mins : 6;
uint8_t seconds : 6; //seconds, max val 59
}time_t;
typedef struct{
uint8_t year : 7; //max val = 99
uint8_t month : 4; //max val = 15
uint8_t day_1 : 5; //max val = 31
uint8_t day_2 : 3; //day_1 - day of month, day_2 - day of week
}date_t;
typedef struct{
date_t date;
time_t time;
}time_data_t;
typedef struct{
uint8_t days_flags; //days of week when alm has to be triggered
uint8_t other_flags : 2; //second snoze, first alm trigerred
}flags_t;
typedef struct{
time_t alm_time;
flags_t flags;
uint32_t tim; //max 7200000 -> 60 min
uint8_t state;
}alarm_t;
extern uint8_t flags;
void check_alarm(alarm_t*, const time_data_t*, uint8_t, const uint8_t);
void init_alarms(alarm_t*, uint8_t);
#endif
```

Plik „data_structs.h”.

Plik zawiera definicje struktur danych potrzebnych do przechowywania takich wielkości jak czas, data, flagi oraz stany alarmów, typu wyliczeniowego, przez który kod sprawdzający maksymalną ilość dni w miesiącu przy nastawach jest bardziej czytelny. W pliku również zostały zawarte: deklaracje funkcji inicjalizującej alarmy, sprawdzającej stan alarmów oraz zadeklarowana została zmienna dostępna globalnie, która przechowuje flagi przydatne w określaniu niektórych stanów systemu takich jak sprawdzenie, czy alarm jest aktywny.

```
#include "data_structs.h"
uint8_t flags;//first bit stands for alarm, second for buzzing while mins == secs == 00, third exit, fourth is set if buzzer enabled
//fifth for check wheter data or time is being set
void check_alarm(alarm_t* alarms, const time_data_t* time, uint8_t quantity, const uint8_t coef){
alarm_t* pt = alarms;
uint8_t tmp = 0;
for(uint8_t i = 0; i < quantity; ++i){
switch(pt[i].state){
case 1:
tmp = (pt[i].alm_time.hours == time->time.hours) && (pt[i].alm_time.mins == time->time.mins) &&
(pt[i].flags.days_flags & (1 << time->date.day_2)) && !time->time.seconds;
if((pt[i].flags.other_flags = tmp? 1 : 0)){
if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
pt[i].state = 2;
}
break;
case 2:
if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
if(okK){
pt[i].tim = 60;
pt[i].state = 3;
}
else if(incK){
pt[i].tim = 60;
pt[i].state = 5;
}
break;
case 3:
if(pt[i].tim && !okK){
pt[i].state = 2;
}
else if(!pt[i].tim && okK){
if(i >= 5){ //5 is a first non repetive alm
pt[i].flags.days_flags = 0;
}
pt[i].flags.other_flags = 0;
pt[i].state = 4;
}
break;
case 4:
if(!okK){
pt[i].state = 1;
}
break;
case 5:
if(pt[i].tim && !incK){
pt[i].state = 2;
```

```
}
else if(!pt[i].tim && incK){
pt[i].flags.other_flags = 2;
pt[i].state = 6;
}
break;
case 6:
if(!incK){
pt[i].tim = (uint32_t)coef * 2000 * 60;
pt[i].state = 7;
}
break;
case 7:
if(pt[i].tim == 0){
pt[i].flags.other_flags = 1;
pt[i].state = 2;
}
break;
}
if(pt[i].tim > 0) --pt[i].tim;
}
for(uint8_t i = 0; i < quantity; ++i){
if(pt[i].flags.other_flags == 1) ++tmp;
}
if(!tmp) flags &= ~ALARM_MASK;
else if(tmp) if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
}
void init_alarms(alarm_t* alm, uint8_t q){
for(uint8_t i = 0; i < q; ++i){
alm[i].tim = alm[i].flags.other_flags = alm[i].flags.days_flags = 0;
alm[i].state = 1;
alm[i].alm_time.hours = alm[i].alm_time.mins = 0;
}
}
```

Plik „data_structs.c”.

Plik zawiera definicje funkcji inicjalizującej alarmy (ustawia wszystkie nastawy na zera, alarmy wyłączone), oraz funkcji, która sprawdza stany alarmów (sprawdzanie czy alarm powinien zostać aktywowany, w razie gdy jest aktywny, czy powinien zostać wyłączony lub wyłączony na czas drzemki i ponownie włączony).

3.2. Skrócony opis pliku z funkcją main.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <util/delay.h>

#include "spi.h"
#include "controls.h"
#include "chars.h"
#include "timers.h"
#include "twi.h"
#include "data_structs.h"

#define T1_BUZZ 240
#define T2_BUZZ 180
#define T1_CONTROLS 200
#define T2_CONTROLS 400
#define NEW_HOUR 0x02
#define EXIT_CONDITION 0x04
#define BUZZ_ENABLED 0x08
#define TURN_BUZZER_ON PORTC &= ~(1 << 0)
#define TURN_BUZZER_OFF PORTC |= (1 << 0)
#define SETTINGS_ON 0x10

volatile uint8_t cycle = 0;
static uint8_t date_buff[7] = {0};
static alarm_t alarms[10];
```

Definicje makr upraszczające sprawdzanie stanów zapisanych w zmiennej flags, zmiennych wykorzystywanych do operacji niskopoziomowych i przechowywania nastaw alarmów oraz załączenie odpowiednich plików nagłówkowych.

```
init_alarms(alarms, 10);
init_timers();
init_spi();
TWBR = 72; //TWI prescaler ~91kHz
init_RTC();
sei();
DDRC = 0x01;
DDRD = 0x7F & ~0x03; //set mux outs, dont interrupt S1 and S2
flags = 0x00;
controls_init(&PORTD, 1, &PORTD, 0); //init controls
function_init(&PORTB, 1, &PORTD, 7);
reg_fn_pins(&PINB, &PIND);
reg_pins(&PIND, &PIND); //register pins for controls
```

Inicjalizacja peryferiów, timera sprzętowego oraz przypisanie 0x00 zmiennej flags.

```
/*date*/
time_data_t matrix_date = {{0, 0, 0, 0}, {0, 0, 0}};
uint8_t date_state = 1;
uint16_t date_tim = 1000;
/*date*/
/*----------display data start----------*/
reg_data_t data;
data.first = 0xff;
data.second = data.third = 0xff;
uint8_t first = 0, second = 0, third = 0, fourth = 0; //chars to be displayed
char dot = 0;
/*----------display data end--------------*/
/*-------- buzzer vars start --------------*/
uint8_t pc_buzz = 0, i_buzz = 0;
uint16_t tim_buzz = 0;
char buz_out, buz_cond = 0;
uint16_t tim_hour_buzz = 0;
uint8_t hour_buzz_state = 1;
/*-------- buzzer vars end ----------------*/
/*---------------------controls------------*/
uint8_t S1, S2;
/*----------controls end--------------------*/
/*--------------main graph vars start------*/
uint8_t S3, S4, pc_main = 0;
char main_out = 0x00, main_cond = 0;
uint16_t tim_main = 0, side_tim = 0, inc_dec_tim = 0;
char main_iter = 0, side_iter = 0, side_state = 1;
/*--------------main graph vars end--------*/

/*--------------alm vars start-------------*/
uint8_t snoze_time_coef = 1, inc_state = 1;
//static alarm_t alarms[0];
/*--------------alm vars end---------------*/
/*multiplexer vars start------------------*/
uint8_t i_mux = 0, mux_state = 1;
/*multiplexer vars end---------------------*/
```

Definicje zmiennych wykorzystywanych przez funkcje main do manipulacji peryferiami, zarządzania wewnętrznymi stanami oraz wewnętrznymi timerami, których baza czasowa opiera się na timerze sprzętowym (przez małą dokładność timera sprzętowego, do zliczania czasu zastosowany został układ RTC).

- krótki opis pętli nieskończonej funkcji main

W cyklach trwających $\frac{1}{2000}s$ wykonywane są wszystkie czynności potrzebne do poprawnego funkcjonowania sterownika. Pierwszym grafem, który determinuje to, w jakim stanie znajduje się system jest „main graph”, który cyklicznie sprawdza pojawienia się warunku przejścia do menu nastaw (tj. wciśnięty przycisk „FUNCTION”), jest to graf binarny, którego tablice warunków skoku, tablice adresów oraz tablice stanów wyjść zawarte są w pamięci EEPROM procesora. Po przejściu do stanu, w którym przeprowadzane są nastawy podstawowe sprawdzany jest stan alarmu, jeśli jest on aktywowany to następuje bezwarunkowe przejście do stanu podstawowego (wyświetlanie godziny i daty). Jeśli zaś alarm jest nieaktywny, po przejściu do którejkolwiek opcji w menu nastaw graf oczekuje na warunek wyjścia („flags & EXIT_CONDITION”) aby powrócić do głównego menu lub na alarm by wrócić do podstawowego stanu.

Następnym grafem jest automat aktualizujący wyświetlaną datę, która pobierana jest z układu RTC a następnie konwertowana na system dziesiętny. Aktualizacja daty dzieje się we wszystkich stanach poza tymi, które wymagają wprowadzania zmian w zliczanej przez RTC zawartości (nastawy czasu i daty).

Kolejnym automatem jest graf sterujący brzęczykiem zależnie od stanów flag. Odpowiedzialny jest za brzęczenie w trakcie alarmu w taki sposób, aby brzmiał jak zwykły budzik.

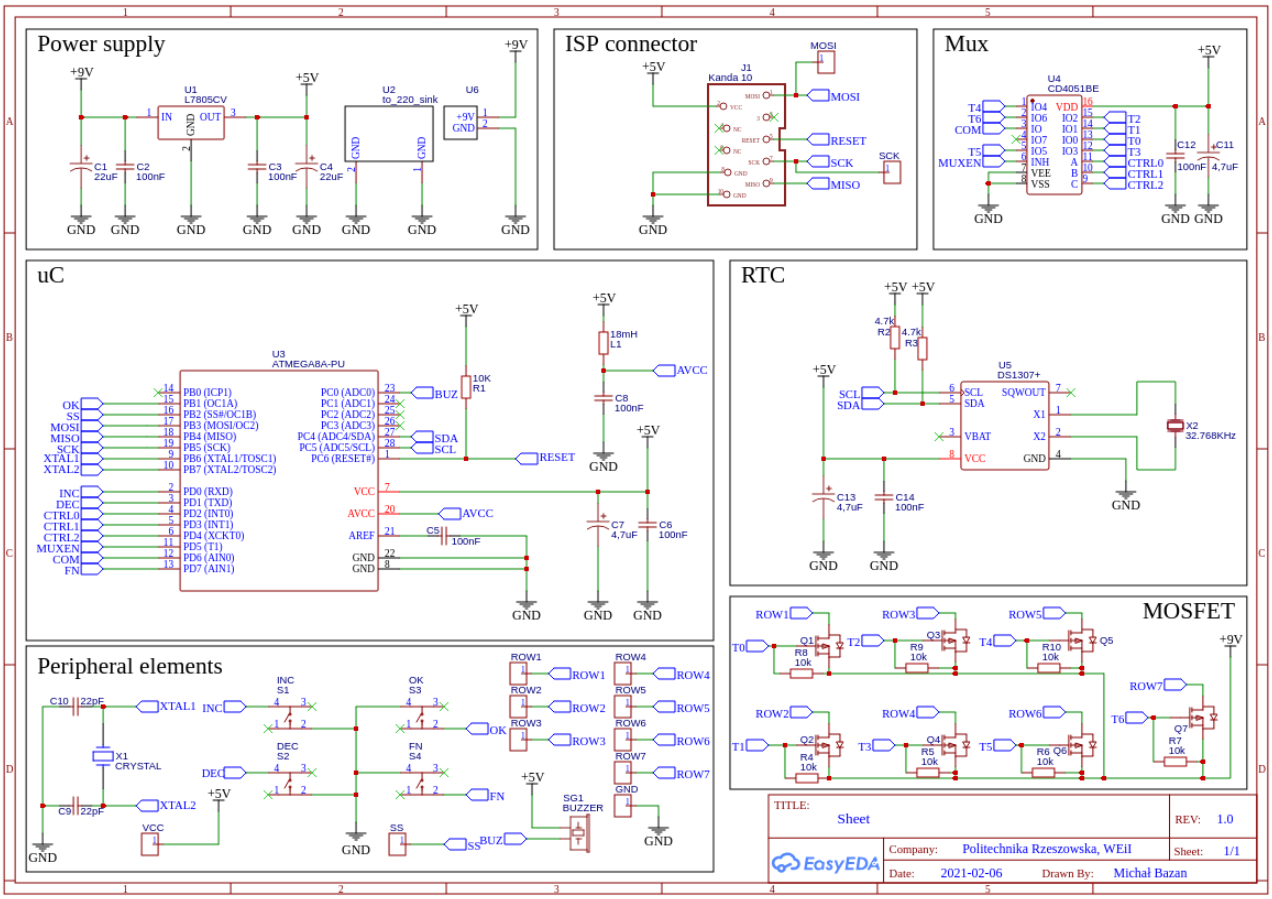
Następny blok instrukcji warunkowych pozwala na sterowanie zawartością wyświetlanej na matrycy LED, zależnie od stanów grafu głównego i ewentualnie podgrafów pozwalających na nastawy podstawowe.

Kolejnymi automatami są grafy, które pozwalają na zmianę czasu, daty, czasu drzemki, nastawy alarmów oraz przełączanie brzęczyka. Wszystkie te grafy mają swoje własne podgrafy, które manipulują zawartością wyświetlaną.

Ostatnim do omówienia jest blok sterujący multiplekserem, który wyłącza wyjścia układu cyfrowego, przełącza aktywne wyjście, przygotowuje zestaw danych dla rejestrów matrycy, przesyła dane oraz włącza multiplekser.

Pod blokiem multipleksera znajdują się instrukcje sterujące „timerami” programowymi oraz pętla, która oczekuje na przerwanie zegarowe timera sprzętowego (otrzymanie cyklu $\frac{1}{2000}s$).

Maksymalne dopuszczalne napięcie zasilania: $U_{Imax} = 9V\ DC$
Średni pobór prądu $0.7\ A | U_I = 9V$



Schemat ideowy sterownika matrycy.

- Power supply

Ta sekcja układu zawiera stabilizator napięcia stałego doprowadzonego do układu poprzez pady oznaczone jako „U6”, zawarty również został radiator odprowadzający ciepło ze stabilizatora.

- ISP connector

Standardowa KANDA pozwalająca na przeprogramowanie procesora oraz wyprowadzenia sygnałów SPI.

- Mux

Sekcja zawiera multiplekser, który pozwala na zasilenie odpowiedniego tranzystora zasilającego pożądany wiersz matrycy LED.

- uC

Ta część układu zawiera mikroprocesor, jego zasilanie oraz wyprowadzenia odpowiednich pinów procesora użytych do sterowania i komunikacji z innymi komponentami.

- RTC

Sekcja zawiera układ DS1307, tj. układ czasu rzeczywistego, który pozwala na relatywnie dokładny pomiar czasu wyświetlanego na matrycy LED. Komunikacja opiera się na interfejsie I^2C . Układ nie ma zasilania bateryjnego.

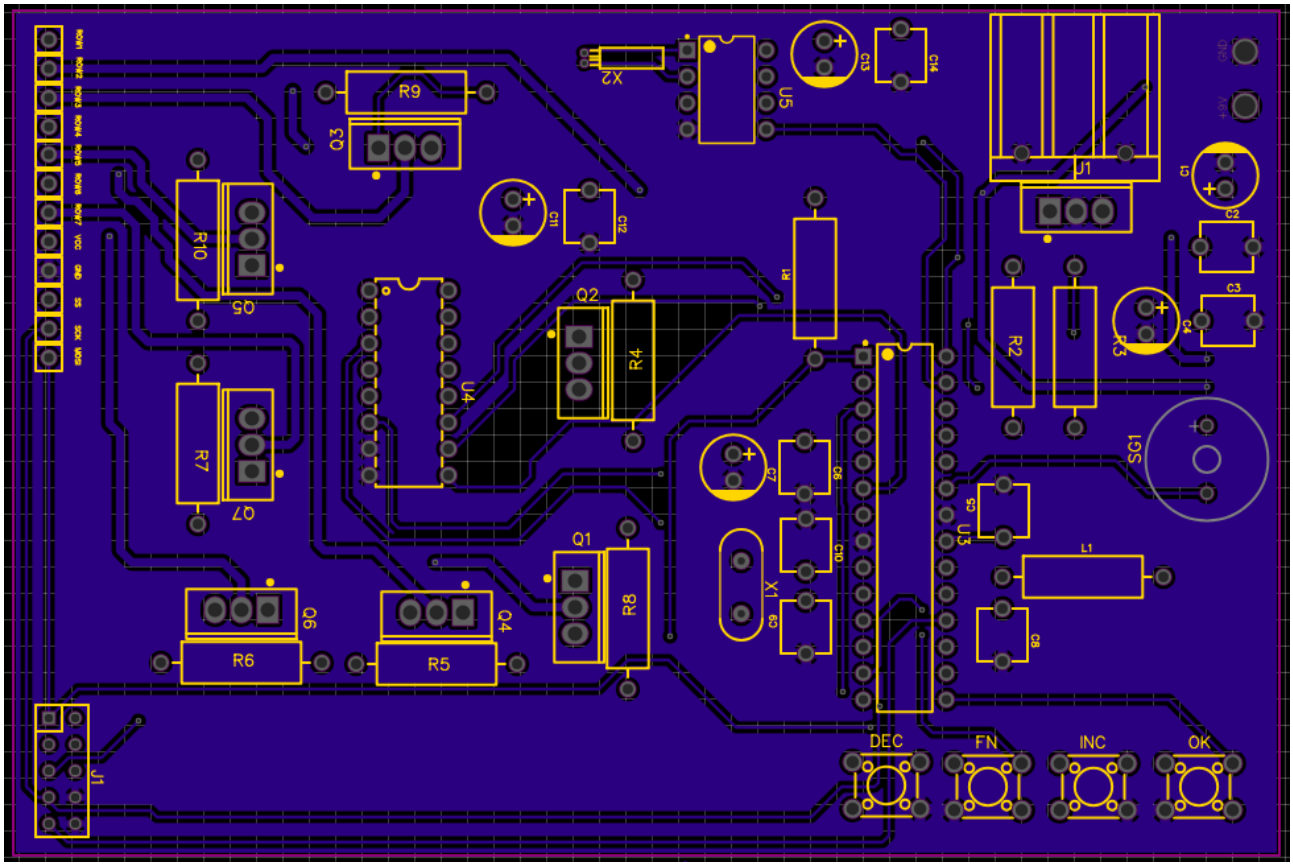
- MOSFET

Ta część układu zawiera tranzystory doprowadzające zasilanie do wierszy matrycy. Są to MOSFETy z kanałem typu P, dlatego aktywowane są stanem niskim dostarczanym przez multiplekser.

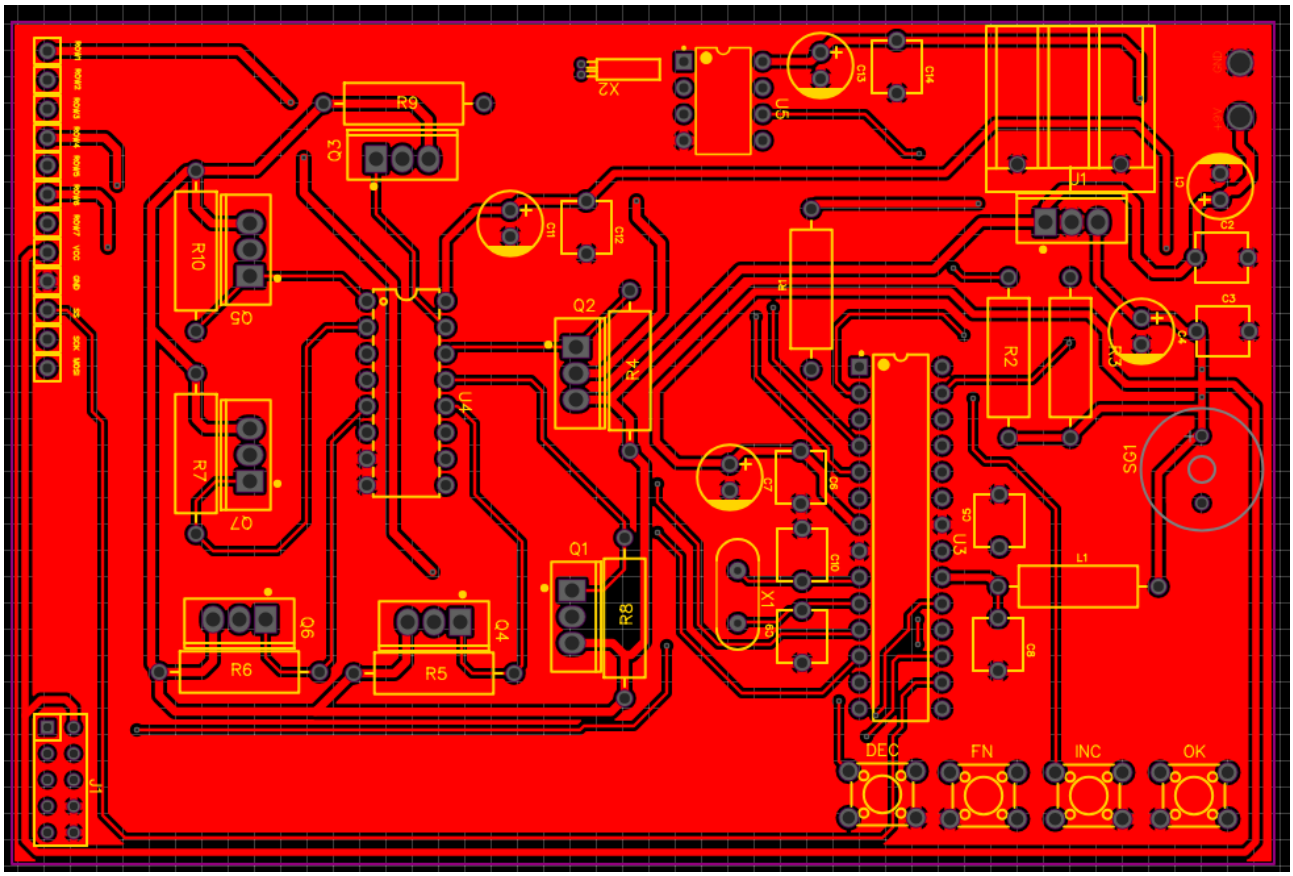
- Peripheral elements

Sekcja zawiera wszystkie elementy, które powinny być kojarzone z doprowadzonym/odprowadzonym sygnałem od/do części wykonawczych układu. W skład wchodzą:

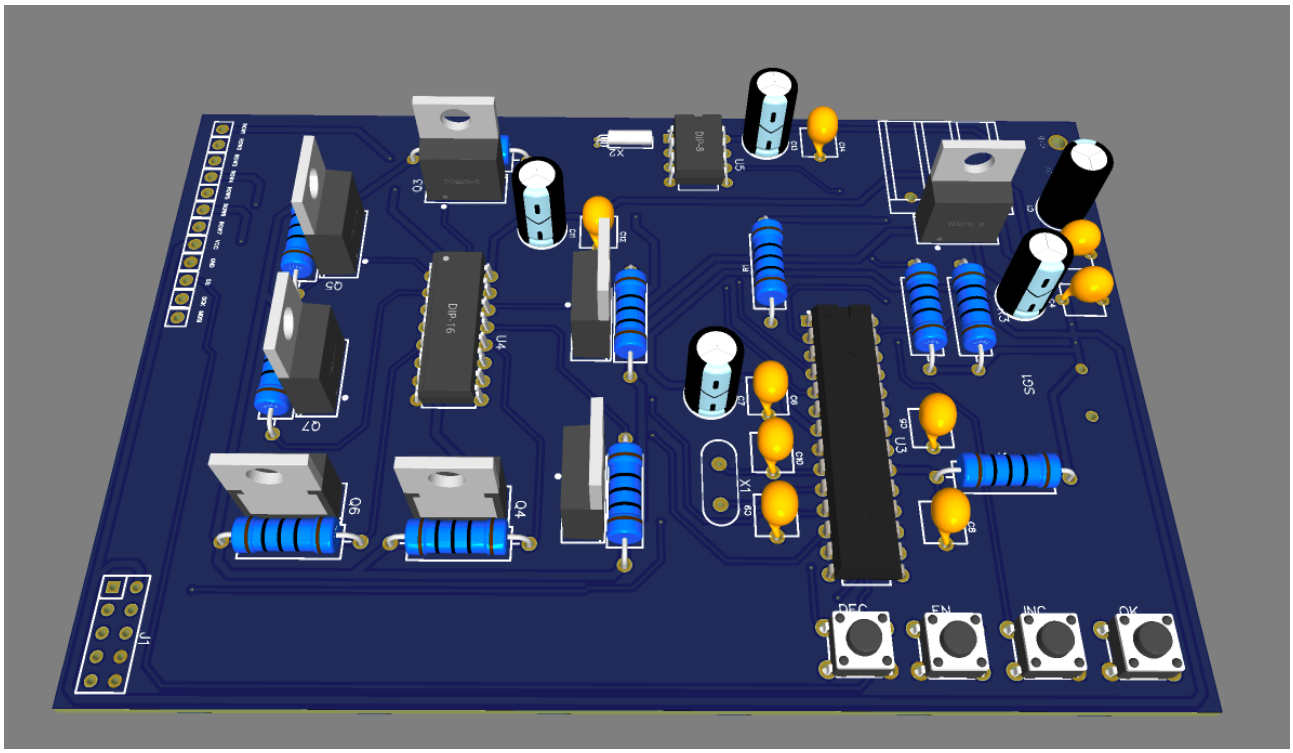
- Wyjścia tranzystorowe zasilające wiersze matrycy oraz wyjście buzzera
- Przyciski wykorzystywane do dokonywania nastaw,
- Taktowanie procesora.



Projekt płytki PCB, połączenia na tylnej stronie.



Projekt płytki PCB, połączenia na przedniej stronie.



Wizualizacja 3D płytki.