Podstawowe parametry:
- obsługa 4 przekaźników sterujących urządzeniami o napięciu sieciowym,
- 10 programów tygodniowych z niezależnym sterowaniem poszczególnych przekaźników lub grupy przekaźników,
- możliwość manualnego sterowania każdym z przekaźników,
- wbudowany zegar czasu rzeczywistego z podtrzymaniem bateryjnym,
- prostota konstrukcji zapewniająca niewielki koszt implementacji,
- łatwość implementacji związana z zastosowaniem elementów do montażu przewlekanego,
- duża ergonomia obsługi,
- czytelne i proste menu.

Dodatkowe materiały do pobrania ze strony www.ulubionykiosk.pl/media

AVT5253	Układ cyklicznego restartu (EP 9/2022)
AVT5948	Wielokrotny włącznik monostabilny (EP 8/2022)
AVT5946	Układ czasowy z niezależną regulacją ON i OFF (EP 8/2022)
----	Wyłącznik czasowy z wejściem bistabilnym (EP 4/2022)
AVT5867	Wyłącznik zasilania z opóźnieniem (EP 6/2021)
AVT5730	Uniwersalny układ czasowy 230 V (EP 11/2019)
AVT5704	Programowany układ czasowy 230 V (EP 8/2019)
AVT5666	Programowany, 16-kanałowy sterownik 230 V (EP 3/2019)
AVT1998	Karta przekaźników programowana sekwencjami (EP 8/2018)
AVT5588	Sterownik-timer z 8 przekaźnikami (EP 6/2017)
AVT5561	Efektowny sterownik oświetlenia (EP 12/2016)
AVT1916	Konfigurowalny przełącznik 4-kanałowy (EP 8/2016)
AVT1890	Moduł przekaźników z USB (EP 6/2016)

W ofercie AVT*
AVT5974

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: **kity@avt.pl**.

Multisterownik

Idea powstania tego projektu nie jest zbyt oryginalna, gdyż jak to często bywa, powstał on z potrzeby chwili. Wspomnianą potrzebą była konieczność cyklicznego sterowania pracą kilku urządzeń o napięciu sieciowym zgodnie z zaprogramowanym harmonogramem tygodniowym. Właśnie wtedy dość nieoczekiwanie zdałem sobie sprawę, że pomimo kilkunastu lat doświadczenia w projektowaniu urządzeń mikroprocesorowych i prawie 150 wdrożonych w życie projektów, nigdy wcześniej nie dane mi było implementować tego rodzaju sterownika, a że lubię zrobić wszystko po swojemu i tym razem dałem upust mojej twórczej wyobraźni. Tak powstał projekt o nazwie multisterownik.

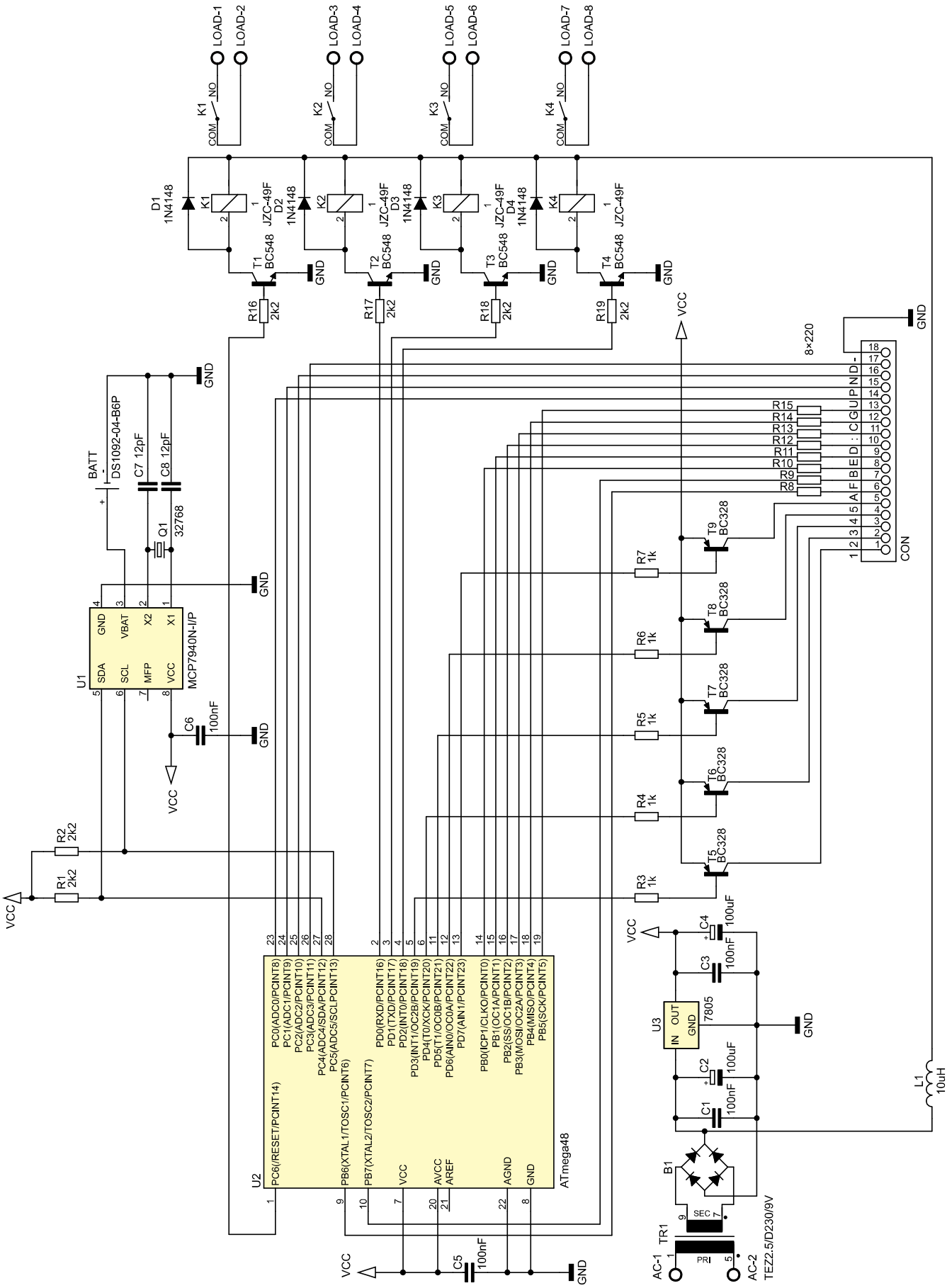
Jednym z założeń projektu była duża ergonomia obsługi, dlatego postanowiłem, że konstrukcja urządzenia będzie przypominała typowy zegar biurkowy, w związku z czym projekt ideowy (jak i montażowy) podzieliłem na 2 części: płytę główną i płytę czołową montowaną pod kątem 90° do płyty głównej a stanowiącą jednocześnie element interfejsu użytkownika. Co oczywiste, na wspomnianej płycie czołowej znajdą się wszystkie elementy sygnalizacyjne typu diody LED czy wyświetlacz LED, jak i sterujące w rodzaju microswitchy.

Budowa i działanie

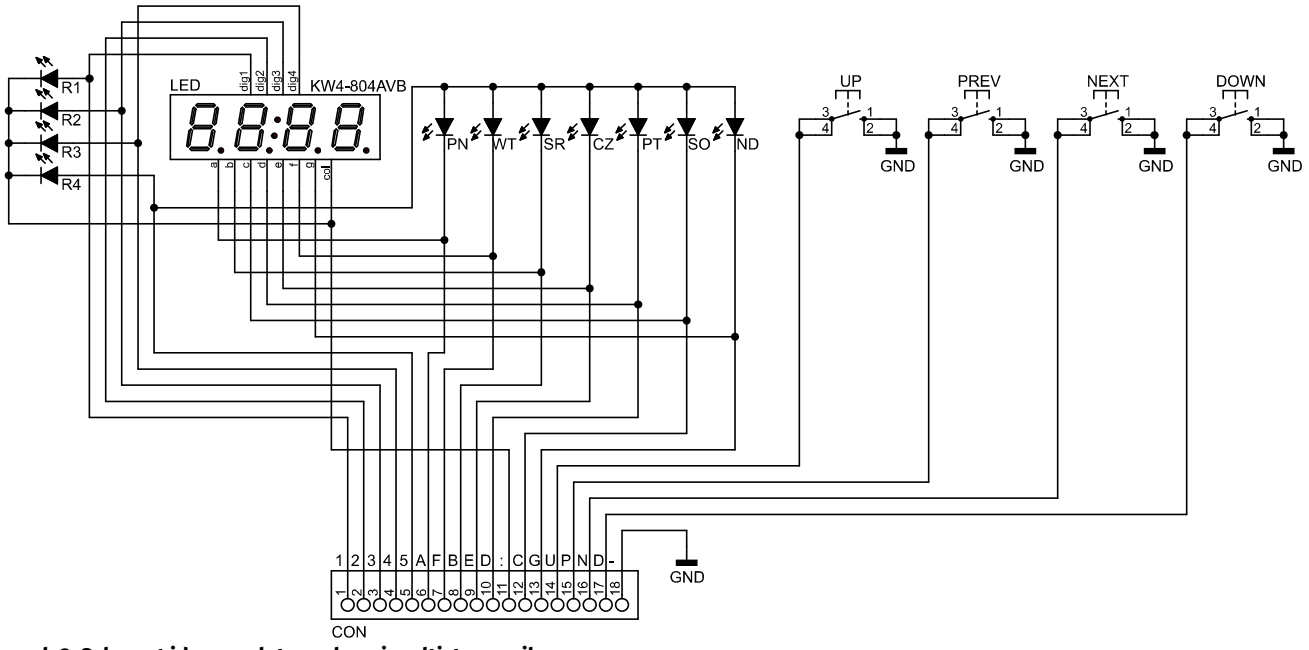
Przejdźmy zatem do schematu ideowego płyty głównej, który został pokazany na **rysunku 1**. Jak widać, zaprojektowano bardzo prosty system mikroprocesorowy, którego sercem jest niewielki mikrokontroler ATmega48 firmy Microchip (dawniej Atmel) taktowany wewnętrznym oscylatorem RC o częstotliwości 1 MHz realizujący całą założoną funkcjonalność urządzenia. Mikrokontroler steruje pracą 7-segmentowego wyświetlacza LED o organizacji 4 znaków w konfiguracji wspólnej anody (wyprowadzenia PD3…PD6 układu), grupą diod LED połączonych w takiej samej konfiguracji wspólnej anody (wyprowadzenie PD7), a stanowiących dodatkowy element interfejsu użytkownika (pokazują dni tygodnia i stan przekaźników sterujących) oraz grupą 4 przycisków sterujących typu microswitch (wyprowadzenia PC0…PC3 układu) przeznaczonych do obsługi urządzenia.

Wspólne anody elementów LED, o których mowa powyżej, sterowane są poprzez proste klucze tranzystorowe T5…T9 z uwagi na dość duże prądy o wartościach rzędu 120 mA (8×15 mA na segment). Wspólne katody wspomnianych powyżej elementów LED obsługiwane są z kolei przez wyprowadzenia PB0…PB7 mikrokontrolera i jak już można się domyślić – do ich obsługi (jak i wspólnych anod) zastosowano doskonale znany mechanizm multipleksowania. Wszystkie wspomniane wcześniej porty sterujące doprowadzono do złącza CON (typu GOLDPIN), przez co umożliwiono łatwe połączenie modułu płyty głównej z modułem płyty czołowej urządzenia.

Mikrokontroler obsługuje ponadto 4 przekaźniki dużej mocy (poprzez proste klucze tranzystorowe T1…T4) oraz zegar czasu rzeczywistego z podtrzymaniem bateryjnym pod postacią układu MCP7940N-I/P firmy Microchip. Obsługa tego układu stała się możliwa dzięki zastosowaniu interfejsu TWI wbudowanego w strukturę mikrokontrolera będącego funkcjonalnym odpowiednikiem



Rysunek 1. Schemat ideowy płyty głównej multisterownika



Rysunek 2. Schemat ideowy płyty czołowej multisterownika

interfejsu I²C firmy Philips. Już teraz zwrócę uwagę, że jeśli chcemy, aby nasz zegar wspierał funkcję podtrzymywania bateryjnego, należy zastosować dokładnie taki typ układu, jaki podano powyżej (i w spisie elementów), gdyż producent tego peryferium oferuje także wersje bez tej funkcjonalności oznaczone innym sufiksem.

Kończąc temat płyty głównej, należy zauważyć, że zaprojektowano tutaj niewielki, kompletny układ zasilający przeznaczony do podłączenia do sieci 230 V. Mimo dość dużych, jak na systemy mikroprocesorowe, prądów sterujących (wyświetlacz LED – 120 mA, przekaźniki 10 mA), maksymalny pobór mocy całego urządzenia nie powinien przekroczyć 2 W.

Przejdźmy zatem do schematu ideowego płyty czołowej, który pokazano na **rysunku 2**. Jak widać, zaprojektowano niezmiernie prosty panel czołowy integrujący 4-znakowy, 7-segmentowy wyświetlacz LED (o wysokości 20,4 mm), grupę 11 dodatkowych diod LED oraz 4 przyciski funkcyjne, zaś wszystkie sygnały sterujące wyprowadzono na złącze CON (typu GOLDPIN). Sposób połączeń wyświetlacza LED i diod LED wynika z zasady sterowania tymi elementami bazującej na mechanizmie multipleksowania, przy tym większość (a dokładnie 8) anod diod LED połączono razem, tworząc niejako kolejną (piątą) wspólną anodę przeznaczoną do wysterowania. Pozostałe trzy połączono do 3 wspólnych anod wyświetlacza LED, uwzględniając fakt, że dwukropek tego wyświetlacza podłączono wyłącznie do drugiej wspólnej anody wspomnianego elementu, przez co niejako „uwolniono" 3 wspólne katody (wyświetlacze 1, 3 i 4), do których właśnie podłączono katody tych 3 diod LED. Brzmi to być może trochę zawile, ale jeśli spojrzymy na schemat ideowy

Listing 1. Plik nagłówkowy mechanizmu multipleksowania

```
//Port katod (segmentów)
#define SEG_DDR DDRB
#define SEG_PORT PORTB

//Definicje konfiguracji poszczególnych segmentów (katod)
#define SEG_A PB6
#define SEG_B PB0
#define SEG_C PB4
#define SEG_D PB2
#define SEG_E PB1
#define SEG_F PB7
#define SEG_G PB5
#define SEG_CL PB3

#define SEG_PN SEG_A
#define SEG_WT SEG_F
#define SEG_SR SEG_B
#define SEG_CZ SEG_E
#define SEG_PT SEG_D
#define SEG_SO SEG_C
#define SEG_ND SEG_G

#define SEG_REL SEG_CL

//Port katod, jako port wyjściowy
#define SEG_AS_OUTPUT SEG_DDR = 0xFF
//Wszystkie segmenty (katody) wygaszone (aktywny stan "0",
//gdyż sterujemy bezpośrednio katodami diod LED)
#define SEG_BLANK SEG_PORT = 0xFF

//Port wspólnych anod – tranzystory sterujące
#define COM_DDR DDRD
#define COM_PORT PORTD

//Definicje konfiguracji poszczególnych wspólnych anod
#define COM_DIG1 PD3
#define COM_DIG2 PD4
#define COM_DIG3 PD5
#define COM_DIG4 PD6
#define COM_DIG5 PD7

//Port wspólnych anod, jako port wyjściowy
#define COM_AS_OUTPUT COM_DDR |=
    (1<<COM_DIG5)|(1<<COM_DIG4)|
    (1<<COM_DIG3)|(1<<COM_DIG2)|(1<<COM_DIG1)
//Wszystkie wspólne anody wyłączone (aktywny stan "0",
//gdyż sterujemy bazami tranzystorów PNP)
#define COM_BLANK COM_PORT |=
    (1<<COM_DIG5)|(1<<COM_DIG4)|
    (1<<COM_DIG3)|(1<<COM_DIG2)|(1<<COM_DIG1)

//Definicje dla Timera1
//Preskaler = 1
#define START_TIMER1 TCCR1B = (1<<WGM12)|(1<<CS10)
#define STOP_TIMER1 TCCR1B = (1<<WGM12)

//Definicja bitu odpowiedzialnego za miganie cyfry
#define BLINKING_BIT 0b10000000

//Indexy znaków specjalnych
#define BLANK_DIGIT_NR 10
#define CHAR_P_NR 11
#define CHAR_F_NR 12
#define CHAR_o_NR 13
#define CHAR_r_NR 14
#define CHAR_d_NR 15
#define CHAR_n_NR 16
#define CHAR_i_NR 17
#define CHAR_b_NR 18
#define CHAR_E_NR 19
#define CHAR_U_NR 20
#define CHAR_DASH_NR 21

//Deklaracje zmiennych globalnych
//Zmienna przechowująca wartość wyświetlaną na wyświetlaczu LED
extern volatile uint8_t Digit[4];
//Zmienna przechowująca wartość wyświetlaną na linijce dni tygodnia
extern volatile uint8_t Day;
//Zmienna przechowująca wartość wyświetlaną na linijce przekaźników
extern volatile uint8_t Relay;
//Wskaźnik załączenia dwukropka
extern volatile uint8_t Semi;
//Wskaźnik aktywnej funkcji przyciemniania
extern volatile uint8_t Dim;
//Zezwolenie na atomową zmianę zmiennych
extern volatile uint8_t readyForUpdate;
```

Listing 2. Definicje niezbędnych stałych mechanizmu multipleksowania

```
//Definicje znaków wyświetlacza LED (aktywny stan "0", gdyż sterujemy bezpośrednio katodami diod LED)
const uint8_t digitPattern[] = {
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)),  //0
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)),  //1
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_G)),  //2
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_G)),  //3
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)|(1<<SEG_F)|(1<<SEG_G)),  //4
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_F)|(1<<SEG_G)),  //5
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //6
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)),  //7
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //8
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_F)|(1<<SEG_G)),  //9
    0xFF,  //Wyświetlacz wygaszony (10)
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //P (11)
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //F (12)
    (uint8_t) ~((1<<SEG_G)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)),  //o (13)
    (uint8_t) ~((1<<SEG_E)|(1<<SEG_G)),  //r (14)
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_G)),  //d (15)
    (uint8_t) ~((1<<SEG_C)|(1<<SEG_E)|(1<<SEG_G)),  //n (16)
    (uint8_t) ~((1<<SEG_C)),  //i (17)
    (uint8_t) ~((1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //b (18)
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //E (19)
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)),  //U (20)
    (uint8_t) ~((1<<SEG_G))  //- (21)
};

//Definicje dni tygodnia (aktywny stan "0",
//gdyż sterujemy bezpośrednio katodami diod LED)
const uint8_t weekDayPattern[] = {
    (uint8_t) ~(1<<SEG_PN),
    (uint8_t) ~(1<<SEG_WT),
    (uint8_t) ~(1<<SEG_SR),
    (uint8_t) ~(1<<SEG_CZ),
    (uint8_t) ~(1<<SEG_PT),
    (uint8_t) ~(1<<SEG_SO),
    (uint8_t) ~(1<<SEG_ND)
};

//Definicje dla portu sterującego wspólnymi anodami wyświetlaczy LED
//(aktywny stan "0", gdyż sterujemy bazami tranzystorów PNP)
const uint8_t comPattern[] = {
    (uint8_t) ~(1<<COM_DIG1),  //Wspólna anoda cyfry 1 (pierwsza z lewej)
    (uint8_t) ~(1<<COM_DIG2),  //Wspólna anoda cyfry 2
    (uint8_t) ~(1<<COM_DIG3),  //Wspólna anoda cyfry 3
    (uint8_t) ~(1<<COM_DIG4),  //Wspólna anoda cyfry 4
    (uint8_t) ~(1<<COM_DIG5),  //Wspólna anoda linijki dni tygodnia
};

//Definicje pozycji bitów zmiennej Relay (dla uproszczenia funkcji ISR)
const uint8_t Bits[] = {0, 0, 1, 2, 3};
```

modułu płyty czołowej i schemat połączeń wewnętrznych zastosowanego wyświetlacza LED, to wszystko stanie się klarowne.

Program sterujący

Tyle w kwestiach konstrukcyjnych, przejdźmy zatem do zagadnień programowych, w ramach których chcę pokazać mechanizm sterowania panelem czołowym, a dokładnie wbudowanym wyświetlaczem LED. Jest to typowe rozwiązanie z użyciem mechanizmu multipleksowania i sekwencyjnym sterowaniem kolejnych cyfr wyświetlacza LED (i linijki dni tygodnia jako 5. wspólnej anody), gdzie przeprowadzamy kolejne i następujące po sobie operacje:

- wyłączamy wszystkie wspólne anody, wyłączając tym samym wszystkie elementy LED,
- na port wspólnych katod wystawiamy wzór do wyświetlenia (aktywny stan 0),
- załączamy wybraną wspólną anodę (aktywny stan 0), wyświetlając tym samym wcześniejszy „wzór" na wybranym elemencie LED (znaku wyświetlacza 7-segmentowego lub linijce LED),
- powtarzamy powyższy proces dla kolejnych wspólnych anod.

Powyższy proces wykonywany dostatecznie szybko (w naszym wypadku 60 razy na sekundę dla każdej wspólnej anody) pozwala na obsłużenie 40 elementów LED (segmentów wyświetlacza 7-segmentowego i dodatkowych diod LED) przy udziale wyłącznie 13 wyprowadzeń mikrokontrolera. Prawda, że proste? A jakie efektywne! Już teraz powiem, że użyjemy do tego celu 2 układów czasowo-licznikowych wbudowanych w strukturę mikrokontrolera: Timer0 i Timer1, przy czym zastosowanie tego drugiego jest opcjonalne.

Timer0 będzie pracował w trybie CTC i będzie wywoływał stosowne przerwanie (od porównania) 300 razy na sekundę (60 razy dla każdej wspólnej anody), obsługując właściwy mechanizm multipleksowania, zaś Timer1 skonfigurowany zostanie w taki sposób, by w razie potrzeby po około 0,4 ms od załączenia wspólnej anody (które następuje w przerwaniu od Timera0) dokonywał jej wyłączenia (w swoim przerwaniu), zmniejszając tym samym wypadkową jasność świecenia każdej z cyfr LED (jak i dodatkowych diod LED). Widać wyraźnie, że jego działanie pozwala na regulację jasności świecenia wyświetlacza, co zostanie zaimplementowane w programie głównym aplikacji.

Przejdźmy zatem do zagadnień implementacyjnych. Mam świadomość, że nie jest to żadne „rocket science" ani rozwiązanie na wskroś uniwersalne, ale chciałem Wam pokazać, jak w efektywny i efektowny sposób „ogarnąć" tego rodzaju zagadnienie programistyczne, czyniąc sam proces programowania niezmiernie przyjemnym. Nieskromnie powiem, że w moim przekonaniu właśnie w ten przejrzysty sposób powinno się konstruować moduły obsługi danych peryferiów, gdyż jakakolwiek modyfikacja sprowadza się wtedy do kosmetycznych i prostych do wykonania zmian.

Na początek plik nagłówkowy mechanizmu multipleksowania, który pokazano na **listingu 1**, a dzięki któremu porządkujemy późniejszy kod źródłowy, czyniąc go bardzo czytelnym, a jednocześnie upraszczamy proces wprowadzania zmian. Plik ten definiuje główne ustawienia sprzętowe i wprowadza niezbędne zmienne.

Jak widać, w ramach pliku nagłówkowego zadeklarowano szereg zmiennych globalnych (typu *volatile* z uwagi na ich użycie w programie głównym, jak i funkcji *ISR*), które przechowują ustawienia poszczególnych elementów wyświetlacza LED. Niemniej jednak już na tym etapie musimy zdefiniować kilka

stałych opisujących wzorce znaków i upraszczających dostęp do portów sterujących, gdyż zależy nam na tym, aby nasza procedura obsługi przerwania multipleksująca wyświetlacz była jak najkrótsza. Definicje te pokazano na **listingu 2**.

Jak widać, definicje, o których mowa powyżej, zostały umieszczone w pamięci RAM mikrokontrolera. Jest to pewnego rodzaju marnotrawstwo, gdyż stałe te z powodzeniem można (a może nawet wypada) umieścić w pamięci Flash mikrokontrolera, aby nie marnować cennej pamięci RAM, zwłaszcza że wartości tych stałych nasz kompilator i tak musi umieścić, a następnie odczytać, właśnie z tej pamięci Flash na starcie programu obsługi aplikacji (bo skąd miałby wziąć te wartości, aby podstawić je pod odpowiednie tablice?). Dokładnie tak postępowałem dotychczas, pisząc oprogramowanie embedded, jednak dostęp do pamięci Flash jest nieco wolniejszy niż odczyt stałych z pamięci RAM (dokładnie 5 taktów zegara zamiast 2), w związku z czym zdecydowałem się na powyższe rozwiązanie, zwłaszcza że użycie pamięci RAM w naszej aplikacji jest na poziomie 15%. Niby niewielki przyrost szybkości, ale zawsze coś. Skądinąd jest to zgodne z podejściem twórców Androida, który charakteryzuje fraza: „dlaczego nieużywana pamięć RAM ma leżeć odłogiem”? Abstrahując już od celowości i sensowności takiego postępowania, brnijmy dalej.

Pora na zaprezentowanie funkcji konfigurującej mechanizm multipleksowania, jak i niezbędne ustawienia sprzętowe, której ciało pokazano na **listingu 3**. Dalej, na **listingu 4** znajduje się funkcja obsługi przerwania od porównania wartości licznika Timer0 z rejestrem porównania OCR0A odpowiedzialną za realizację mechanizmu multipleksowania wyświetlacza, w której implementacji wzięto pod uwagę obsługę migania elementów LED panelu czołowego. I na sam koniec, na **listingu 5**, mamy funkcję obsługi przerwania od porównania wartości licznika Timer1 z rejestrem porównania OCR1A odpowiedzialną za realizację mechanizmu przyciemniania wyświetlacza LED.

Prawda, że proste? Niemniej jednak warto choćby na chwilę zatrzymać się nad znaczeniem nieopisanej wcześniej zmiennej *readyForUpdate*. Jest to zmienna, która funkcji głównej aplikacji użytkownika wskazuje moment atomowej aktualizacji zmiennych *volatile* procedury obsługi przerwania mechanizmu multipleksowania. Potrzeba

Ustawienia fuse-bitów:
CKSEL3...0: 0010
SUT1...0: 10
CKDIV8: 0
CKOUT: 1
EESAVE: 0
RSTDISBL: 0 (szczegóły w artykule)

Listing 3. Funkcja konfigurująca mechanizm multipleksowania

```
void initMultiplex(void){
    //Porty wspólnych anod i katod,
    //jako wyjściowe ze stanami nieaktywnymi na wyjściach
    SEG_BLANK;
    SEG_AS_OUTPUT;
    COM_BLANK;
    COM_AS_OUTPUT;

    //Konfiguracja układu Timer0 w celu generowania przerwania
    //do obsługi multipleksowania wyświetlacza LED (300 Hz)
    //Tryb CTC
    TCCR0A = (1<<WGM01);
    //Preskaler = 64
    TCCR0B = (1<<CS01)|(1<<CS00);
    //300 Hz (co 3.333 ms)
    OCR0A = 51;
    //Uruchomienie przerwania Timer0 Output Compare Match A
    TIMSK0 = (1<<OCIE0A);

    //Konfiguracja układu Timer1 w celu generowania przerwania
    //po czasie 0,4 ms – do obsługi funkcji przyciemniania wyświetlacza LED
    TCCR1B = (1<<WGM12);  //Tryb CTC
    //Przerwanie po 0,4 ms przy f = 1 MHz i Preskalerze = 1
    OCR1A = 400;
    //Uruchomienie przerwania Timer1 Output Compare Match A
    TIMSK1 = (1<<OCIE1A);
}
```

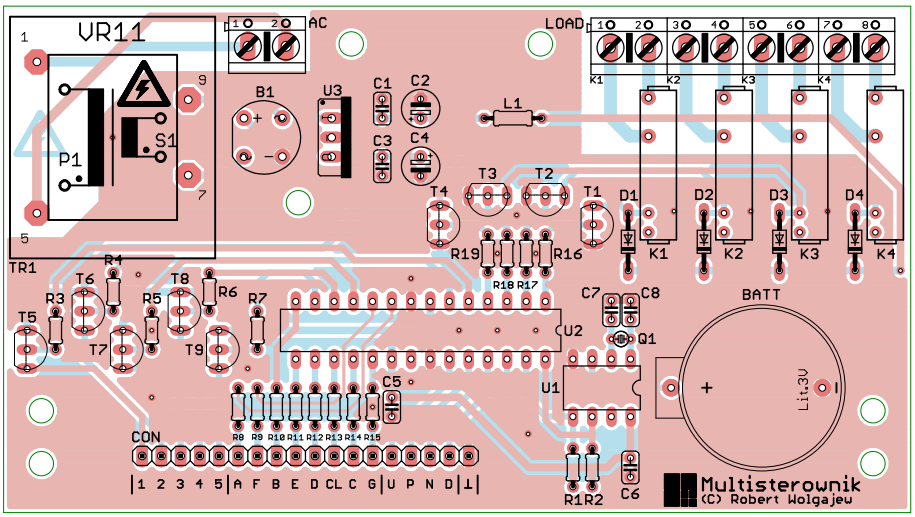
Listing 4. Funkcja obsługi przerwania realizująca mechanizm multipleksowania

```
//Przerwanie obsługi wyświetlacza LED wywoływane co 3,3 ms
//(60 razy na sekundę dla każdej z cyfr/linijek LED)
ISR(TIMER0_COMPA_vect){
    //Numer kolejnej cyfry przeznaczonej do wyświetlenia
    static uint8_t Nr;
    //Timer programowy 3.3ms
    //służący do obsługi migania cyfr wyświetlacza LED
    static uint8_t timer3ms;
    uint8_t segPattern;

    readyForUpdate = 0;
    //Cyfry
    if(Nr < 4){
        if(Digit[Nr] & BLINKING_BIT){
            if(timer3ms & 0x40)
                segPattern = digitPattern[Digit[Nr] & (~BLINKING_BIT)];
            else segPattern = 0xFF;  //Migamy co 211 ms
        }
        else segPattern = digitPattern[Digit[Nr]];
    //Linijka dni tygodnia
    } else {
        //Tworzymy wzór dni tygodnia do wyświetlenia
        segPattern = 0xFF;
        for(uint8_t i=0; i<7; i++)
            if(Day & (1<<i))
                segPattern &= weekDayPattern[i];
        if(Day & BLINKING_BIT)
            if((timer3ms & 0x40) == 0)
                segPattern = 0xFF;          //Migamy co 211ms
    }

    //Zapalamy dwukropek lub diodę przekaźnika (z obsługą migania)
    if(Nr == 1){
        if(Semi)
            segPattern &= ~(1<<SEG_CL);     //Zapalamy dwukropek
    } else {
        if(Relay & BLINKING_BIT){
            if(timer3ms & 0x40)
                if(Relay & (1<<Bits[Nr]))
                    //Migamy co 211ms
                    segPattern &= ~(1<<SEG_REL);
        }
        else if(Relay & (1<<Bits[Nr]))
            segPattern &= ~(1<<SEG_REL);
    }

    //Wyłączenie wspólnych anod wyświetlaczy LED
    COM_BLANK;
    //Wystawienie wzoru na port katod
    SEG_PORT = segPattern;
    //Włączenie odpowiedniej wspólnej anody (aktywny stan “0”)
    COM_PORT &= comPattern[Nr];

    if(++Nr > 4){
        Nr = 0;  //Kolejna wspólna anoda
        //Zezwolenie na atomową zmianę zmiannych w funkcji Main
        readyForUpdate = 1;
    }

    //Obsługa funkcji przyciemniania wyświetlacza LED
    //załączenie taktowania Timera1
    if(Dim)
        START_TIMER1;

    timer3ms++;
}
```

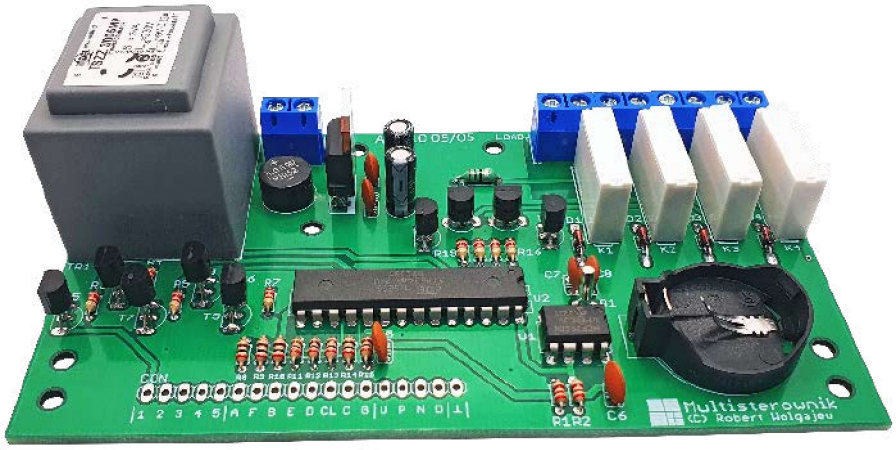
Listing 5. Funkcja przerwania realizująca mechanizm przyciemniania wyświetlacza LED

```
//Przerwanie obsługi funkcji przyciemniania wyświetlacza LED
//wywoływane po 0.4 ms od startu Timera1

ISR(TIMER1_COMPA_vect){
    STOP_TIMER1;  //Wyłączenie taktowania Timera1
    COM_BLANK;  //Wyłączenie wspólnych anod wyświetlaczy LED
}
```

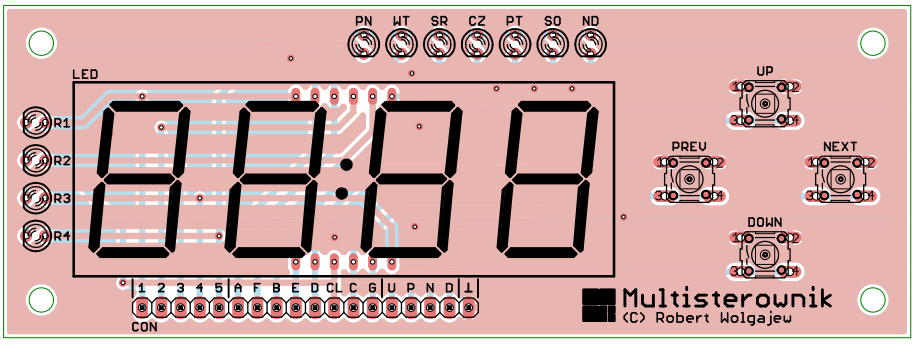
wprowadzenia takiej zmiennej wynikała z konieczności synchronizacji chwili aktualizacji zmiennych dokonywanej w aplikacji głównej z pracą funkcji multipleksującej wyświetlacz LED tak, by nie występowało zjawisko mieszania zawartości zmiennych dla kolejnych przebiegów funkcji multipleksującej. Aktualizacja, o której mowa powyżej, następuje po pełnym cyklu multipleksu dla całego wyświetlacza LED. Tyle w kwestiach implementacyjnych.



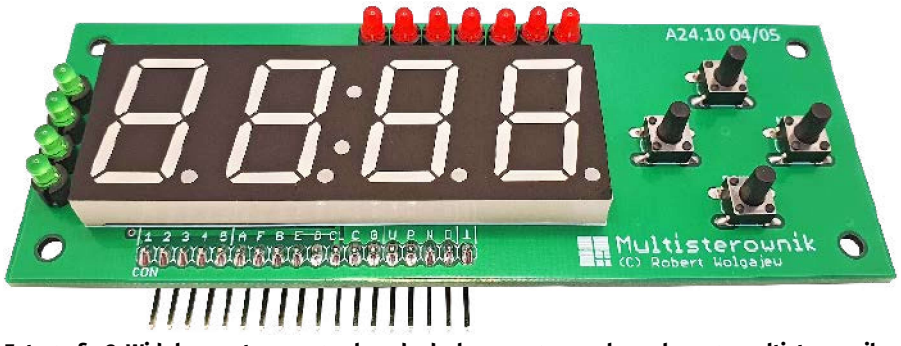
Rysunek 3. Schemat głównej płytki PCB multisterownika



Fotografia 1. Widok zmontowanego obwodu drukowanego płyty głównej multisterownika



Rysunek 4. Schemat płytki PCB panelu czołowego multisterownika



Fotografia 2. Widok zmontowanego obwodu drukowanego panelu czołowego multisterownika

Montaż i uruchomienie

Przejdźmy zatem to schematu montażowego płyty głównej multisterownika, który pokazano na **rysunku 3**. Jak widać, zaprojektowano niewielki obwód drukowany, który integruje w sobie kompletny, transformatorowy zasilacz sieciowy i zbudowany jest wyłącznie z elementów przewlekanych, co w zamyśle miało uprościć montaż urządzenia nawet przez niedoświadczonych amatorów. Montaż obwodu drukowanego płyty głównej multisterownika rozpoczynamy od przylutowania półprzewodników (układy scalone najlepiej zamontować w stosownych podstawkach), następnie lutujemy elementy bierne, dalej przekaźniki, potem gniazdo baterii CR2032, zaś na samym końcu elementy mechaniczne typu złącza LOAD i AC (bez złącza CON) oraz transformator do druku.

Poprawnie zmontowany układ nie wymaga żadnych regulacji i powinien działać tuż po włączeniu napięcia zasilającego (oczywiście po podłączeniu panelu czołowego). Co oczywiste, aby zachować funkcjonalność podtrzymania bateryjnego zegara czasu rzeczywistego, w podstawce baterii podtrzymującej musimy zamontować dowolną baterię typu CR2032. Pobór prądu z tej baterii w czasie braku napięcia zasilającego multisterownik jest rzędu 925 nA, czyli trochę większy od prądu samorozładowania takowej baterii co powinno zapewnić 10 lat podtrzymania pracy zegara.

Oczywiście sam sterownik w przypadku braku napięcia zasilającego nie pracuje, ale pamięta wszystkie ustawienia programów sterujących, jak i podtrzymuje działanie zegara czasu rzeczywistego. Widok zmontowanego obwodu drukowanego płyty głównej multisterownika pokazano na **fotografii 1**.

Uwaga! Na płycie głównej multisterownika zamontowano kompletny zasilacz łącznie z transformatorem zasilanym napięciem sieciowym 230 V AC oraz zamontowano elementy będące na potencjale tego napięcia. Istnieje niebezpieczeństwo porażenia prądem elektrycznym o napięciu 230 V AC, co może stanowić zagrożenie dla życia i zdrowia użytkowników. W związku z tym, montaż układu w tym zakresie należy wykonać pod nadzorem osoby mającej niezbędną wiedzę i doświadczenie.

W tym momencie przechodzimy do schematu montażowego panelu czołowego multisterownika, który to pokazano na **rysunku 4**. Tym razem mamy do czynienia z bardzo prostym, dwustronnym obwodem drukowanym z wyłącznym montażem elementów przewlekanych. Montaż obwodu rozpoczynamy

Wykaz elementów, kupuj na stronie sklep.avt.pl (Warszawa, ul. Leszczynowa 11, tel. +48222578451, e-mail: handlowy@avt.pl)

Rezystory: (obudowy miniaturowe 1/8 W, raster 0,2")
R1, R2, R16...R19: 2,2 kΩ
R3...R7: 1 kΩ
R8...R15: 220 Ω

Kondensatory:
C1, C3, C5, C6: ceramiczny 100 nF (raster 0,1")
C2, C4: elektrolityczny 100 µF/16 V (5 mm, raster 2,5 mm)
C7, C8: ceramiczny 12 pF (raster 0,1")

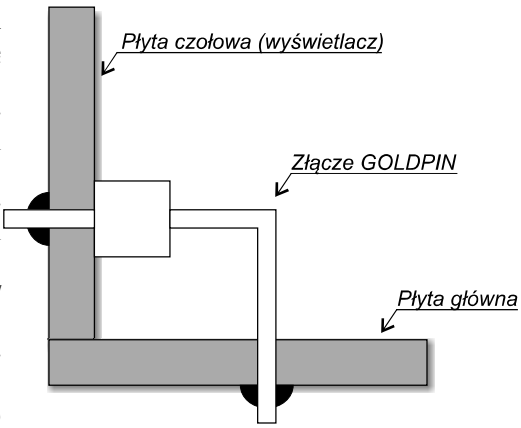
Półprzewodniki:
U1: MCP7940N-I/P (DIL-8)
U2: ATmega48 (DIL-28)
U3: 7805 (TO-220)
T1...T4: BC548 (TO-92)
T5...T9: BC328 (TO-92)
D1...D4: 1N4148 (DO34-7)
B1: mostek prostowniczy 1 A (okrągły, raster 0,2")
R1, R2, R3, R4: dioda LED 3 mm zielona
PN, WT, SR, CZ, PT, SO, ND: dioda LED 3 mm czerwona
LED: wyświetlacz LED 7-segmentowy 20,4 mm typu KW4-804AVB (czerwony, wspólna anoda)

Pozostałe:
Q1: rezonator kwarcowy zegarkowy 32768 Hz
L1: dławik osiowy 10 µH (raster 0,2")
BATT: gniazdo baterii CR2032 do druku typu DS1092-04-B6P
TR1: transformator do druku TEZ2.5/D230/9V
K1...K4: przekaźnik AZ921-1A-12DE ZETTLER (JZC-49F-12V)
UP, DOWN, PREV, NEXT: microswitch 6 mm do druku (9,5 mm)
CON: listwa kołkowa 1×20 kątowa
AC: złącze śrubowe AK500/2
LOAD: złącze śrubowe AK500/8

Opcjonalnie: Tulejka dystansowa LED typu 8GE04V80548 DREMEC (zewn.: 4,8 mm, LED: 3 mm), 11 szt.

od wlutowania wyświetlacza LED, następnie montujemy wszystkie pojedyncze diody LED, zachowując odpowiednią polaryzację i odległość podstawy diod od płaszczyzny obwodu drukowanego (najlepiej użyć specjalnych plastikowych dystansów o wysokości 4 mm), a na końcu switche UP, DOWN, PREV i NEXT oraz złącze kątowe CON typu GOLDPIN, montując je koniecznie od strony warstwy BOTTOM. Widok zmontowanego obwodu drukowanego panelu czołowego multisterownika pokazano na **fotografii 2**.

Tak przygotowaną płytkę panelu czołowego wlutowujemy pod kątem prostym w płytkę płyty głównej, posiłkując się złączem GOLDPIN ustalającym niejako pozycję obu obwodów drukowanych w stosunku do siebie. Sposób połączenia obu obwodów drukowanych multisterownika pokazano na **rysunku 5**. Dla wzmocnienia połączenia obu obwodów drukowanych przewidziano możliwość wzajemnego ich mechanicznego zespolenia za pomocą dwóch prostych kątowników i przygotowanych w tym celu otworów montażowych (zarówno w obwodzie drukowanym płyty głównej, jak i czołowej). Wygląd wspomnianych kątowników pokazano na **rysunku 6**.



Rysunek 5. Sposób połączenia obwodów drukowanych multisterownika względem siebie

Obsługa urządzenia

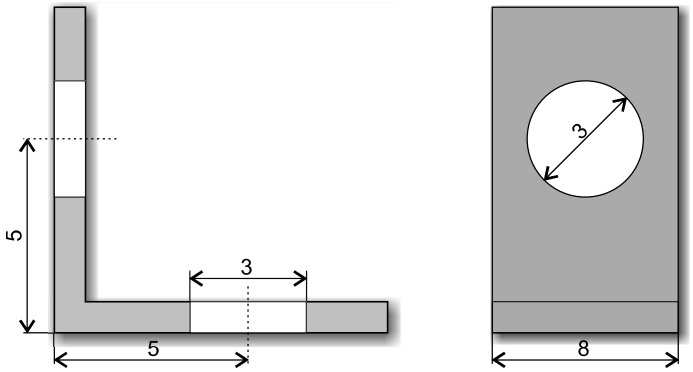
Przejdźmy zatem do obsługi urządzenia. Projektując interfejs użytkownika oraz sposób jego obsługi, przyjąłem, że ergonomia, prostota użytkowania i czytelność interfejsu powinny być najważniejszym kryterium konstrukcyjnym zarówno przy tworzeniu panelu czołowego urządzenia, jak i konstruowaniu stosownych procedur sterujących. Zgodnie z tymi podstawowymi założeniami, na płytce modułu czołowego przewidziano oprócz wyświetlacza 7-segmentowego 7 diod LED przeznaczonych do wyświetlania informacji o dniu (lub dniach) tygodnia, 4 diody LED dające podgląd na stan przekaźników (lub ich konfiguracji w przypadku programów sterujących) oraz 4 przyciski sterujące dające bezpośredni dostęp do podstawowej funkcjonalności.

Wygląd interfejsu użytkownika urządzenia multisterownik pokazano na **rysunku 7**. Menu obsługi urządzenia udostępnia wiele funkcji, dlatego przyciski sterujące i diody LED mają różnorakie znaczenie zależne od miejsca w układzie Menu, przy czym ich podstawowa funkcjonalność prezentuje się następująco:

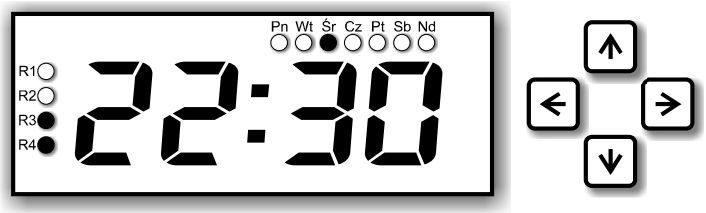
- długie przyciśnięcie przycisku ↑ wprowadza do wyboru i edycji programu sterującego,
- długie przyciśnięcie przycisku ↓ wprowadza do ustawień zegara czasu rzeczywistego,
- długie przyciśnięcie przycisku → włącza/wyłącza funkcję auto-dimmera,
- długie przyciśnięcie przycisku ← wprowadza do trybu manualnego,
- krótkie przyciśnięcie przycisków ← lub → służy podstawowo do zmiany typu edytowanego elementu,
- krótkie przyciśnięcie przycisków ↑ lub ↓ służy podstawowo do zmiany wartości edytowanego elementu,
- przytrzymanie przycisków ↑ lub ↓ służy do szybkiej zmiany wartości edytowanego elementu.

Warto również podkreślić, że w trakcie edycji jakiegokolwiek elementu interfejsu użytkownika jego wartość (tj. jego reprezentacja na wyświetlaczu LED lub na dodatkowych diodach LED) będzie cyklicznie migać, sygnalizując użytkownikowi aktywny proces edycji. Dodatkowo, w trakcie edycji jakiejkolwiek wartości numerycznej przytrzymanie przycisków ↑ lub ↓ powoduje szybkie cykliczne zmiany tej wartości, przyspieszając tenże proces edycji.

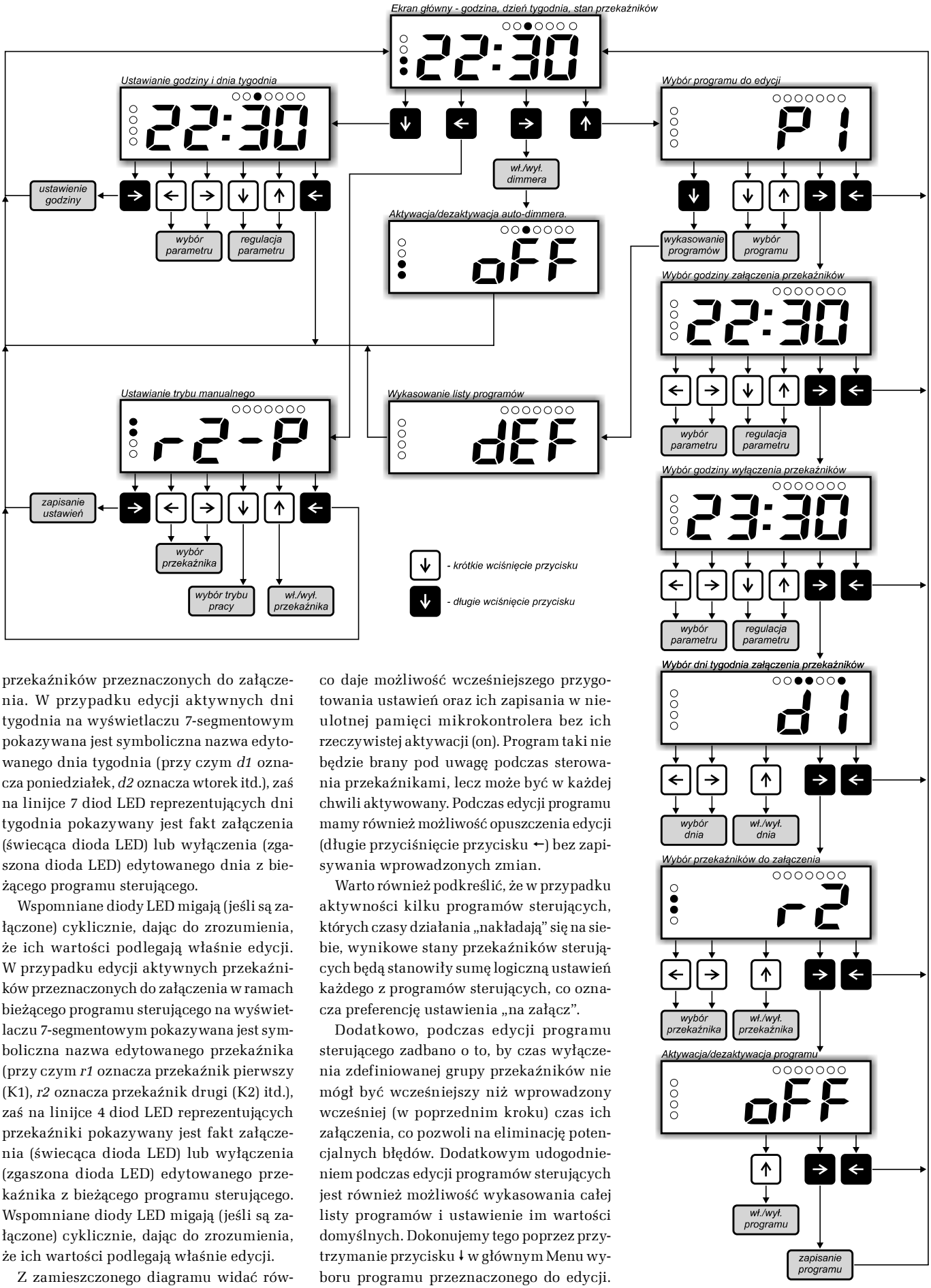
Ponieważ lista dostępnych opcji systemu Menu urządzenia multisterownik jest dość obszerna, na **rysunku 8** pokazano diagram prezentujący kompletny algorytm obsługi. Kilka słów wyjaśnienia wymaga edycja programów sterujących w zakresie aktywnych dni tygodnia i aktywnych



Rysunek 6. Rysunek techniczny kątowników przeznaczonych do połączenia obwodów drukowanych płyty głównej i czołowej multisterownika



Rysunek 7. Wygląd interfejsu użytkownika urządzenia multisterownik



Rysunek 8. Diagram prezentujący kompletny algorytm obsługi urządzenia multisterownik

przekaźników przeznaczonych do załączenia. W przypadku edycji aktywnych dni tygodnia na wyświetlaczu 7-segmentowym pokazywana jest symboliczna nazwa edytowanego dnia tygodnia (przy czym *d1* oznacza poniedziałek, *d2* oznacza wtorek itd.), zaś na linijce 7 diod LED reprezentujących dni tygodnia pokazywany jest fakt załączenia (świecąca dioda LED) lub wyłączenia (zgaszona dioda LED) edytowanego dnia z bieżącego programu sterującego.

Wspomniane diody LED migają (jeśli są załączone) cyklicznie, dając do zrozumienia, że ich wartości podlegają właśnie edycji. W przypadku edycji aktywnych przekaźników przeznaczonych do załączenia w ramach bieżącego programu sterującego na wyświetlaczu 7-segmentowym pokazywana jest symboliczna nazwa edytowanego przekaźnika (przy czym *r1* oznacza przekaźnik pierwszy (K1), *r2* oznacza przekaźnik drugi (K2) itd.), zaś na linijce 4 diod LED reprezentujących przekaźniki pokazywany jest fakt załączenia (świecąca dioda LED) lub wyłączenia (zgaszona dioda LED) edytowanego przekaźnika z bieżącego programu sterującego. Wspomniane diody LED migają (jeśli są załączone) cyklicznie, dając do zrozumienia, że ich wartości podlegają właśnie edycji.

Z zamieszczonego diagramu widać również, że każdy z 10 dostępnych programów sterujących (oznaczony jako P0…P9) możemy włączyć (on) lub wyłączyć (off), co daje możliwość wcześniejszego przygotowania ustawień oraz ich zapisania w nieulotnej pamięci mikrokontrolera bez ich rzeczywistej aktywacji (on). Program taki nie będzie brany pod uwagę podczas sterowania przekaźnikami, lecz może być w każdej chwili aktywowany. Podczas edycji programu mamy również możliwość opuszczenia edycji (długie przyciśnięcie przycisku ←) bez zapisywania wprowadzonych zmian.

Warto również podkreślić, że w przypadku aktywności kilku programów sterujących, których czasy działania „nakładają" się na siebie, wynikowe stany przekaźników sterujących będą stanowiły sumę logiczną ustawień każdego z programów sterujących, co oznacza preferencję ustawienia „na załącz".

Dodatkowo, podczas edycji programu sterującego zadbano o to, by czas wyłączenia zdefiniowanej grupy przekaźników nie mógł być wcześniejszy niż wprowadzony wcześniej (w poprzednim kroku) czas ich załączenia, co pozwoli na eliminację potencjalnych błędów. Dodatkowym udogodnieniem podczas edycji programów sterujących jest również możliwość wykasowania całej listy programów i ustawienie im wartości domyślnych. Dokonujemy tego poprzez przytrzymanie przycisku ↓ w głównym Menu wyboru programu przeznaczonego do edycji. Wspomniane wartości domyślne przedstawiają się następująco:

- godzina załączenia: 12.00,

- godzina wyłączenia: 13.00,
- zdezaktywowane wszystkie dni tygodnia,
- zdezaktywowane wszystkie przekaźniki,
- program nieaktywny.

Tryb manualny

Kilka słów uwagi należy poświęcić również trybowi manualnemu, o którym wspomniano powyżej. Przyznam szczerze, że początkowo nie przewidywałem wprowadzania wspomnianej funkcjonalności, ale jak się szybko okazało, pierwsi użytkownicy urządzenia oczekiwali możliwości manualnego sterowania stanem przekaźników sterujących niezależnie od zaprogramowanego programu (jednego z dziesięciu). Jak się domyślacie, tryb manualny ma pierwszeństwo nad ustawieniami programów sterujących (P0…P9), co oznacza, że w przypadku aktywnego trybu manualnego dla wybranego przekaźnika (konfigurowalne niezależnie dla każdego przekaźnika) jego stan zależny będzie wyłącznie od ustawień trybu manualnego z pominięciem ustawień wszystkich programów sterujących, które ingerują w jego stan.

Po wejściu w tryb manualny na wyświetlaczu LED pokazywany jest numer konfigurowanego przekaźnika (przy czym *r1* oznacza przekaźnik pierwszy (K1), *r2* oznacza przekaźnik drugi (K2) itd.) oraz tryb jego pracy (P → sterowany przez programy sterujące, U → tryb użytkownika (manualny)), zaś na diodach LED (R1…R4) pokazywany jest manualny stan każdego z przekaźników, który stanie się aktywny po włączeniu trybu manualnego dla wybranego przekaźnika.

Co oczywiste, w ramach ekranu głównego (zegara) na wspomnianych diodach LED pokazywany jest wynikowy stan przekaźników sterujących będący kompilacją ustawień programów, jak i trybu manualnego, co znaczy ni mniej, ni więcej, że pokazywany jest faktyczny stan pracy przekaźników.

Ostatnią funkcjonalnością, o jakiej należy wspomnieć i którą można aktywować/dezaktywować z systemu Menu, jest auto-dimmer, czyli automatyczny ściemniacz panelu LED aktywujący się automatycznie w godzinach nocnych tj. od 21.00 do 6.00. Jego zastosowanie ma sens w przypadku, gdy nasz sterownik znajduje się w tym samym pomieszczeniu, w którym śpimy i nie chcemy, by praca wyświetlacza LED zakłócała w jakikolwiek sposób nasz sen. W takim wypadku wyświetlacz LED (jako całość) zostaje zdecydowanie przygaszony.

Programowanie mikrokontrolera

Na koniec słowo na temat programowania mikrokontrolera. Uważny Czytelnik dostrzeże z pewnością, że jeden z przekaźników sterujących (a dokładnie K1) sterowany jest z portu PC6 mikrokontrolera będącego jednocześnie portem sygnału RESET, który domyślnie nie może pełnić funkcję typowego portu I/O. Aby taką funkcję pełnił, niezbędne jest ustawienie (wyzerowanie) fuse-bitu RSTDISBL. Jednak ustawienie (wyzerowanie) tego bitu uniemożliwi dalsze programowanie mikrokontrolera za pomocą zwykłego szeregowego programatora, a jedyną dostępną wtedy opcją stanie się wysokonapięciowy programator równoległy. Wynika z tego, że w pierwszej kolejności należy wgrać oprogramowanie sterujące (HEX), a dopiero później ustawić docelowe fuse-bity. **Uwaga:** brak ustawienia (wyzerowania) bitu RSTDISBL uniemożliwi sterowanie przekaźnikiem K1. Warto również zaznaczyć, że w trakcie programowania port RESET mikrokontrolera powinien być „odpięty” od reszty urządzenia (najprościej nie montować w tym czasie rezystora R16).

Robert Wołgajew, EP