W ofercie AVT*

AVT6043

Najważniejsze parametry:
- Napięcie zasilania: 7...9 V
- Prąd obciążenia: 100 mA
- Źródło napięcia podtrzymania zegara RTC: bateria CR1220
- Prąd podtrzymania zegara RTC: 1 µA
- Zakres pomiarowy wbudowanego termometru: 0...55°C
- Dokładność pomiaru temperatury: 0,5°C
- Rozdzielczość pomiaru temperatury: 0,5/0,1°C (w zależności od rodzaju zastosowanego termometru scalonego)

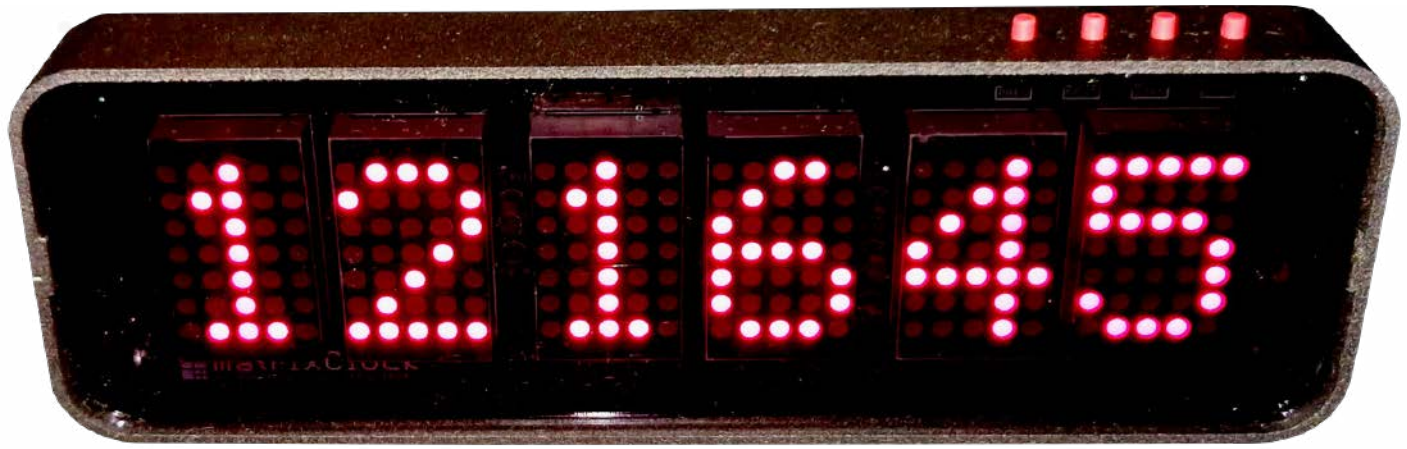
* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie 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**.



matrixClock - efektowny zegar stołowy

Zegary cyfrowe, tak jak termometry, termostaty czy miniaturowe radyjka, należą do elementarza każdego elektronika amatora. Któż z nas nie ma w swoim portfolio podobnych urządzeń, które – mimo oczywistej prostoty – przynoszą dużo radości z własnoręcznej konstrukcji. Również i ja na liście zarówno skonstruowanych, jak i zaprojektowanych przez siebie urządzeń mam kilka takich systemów cechujących się różnym stopniem skomplikowania. I mimo takiego stanu rzeczy postanowiłem ponownie powrócić do tego zagadnienia, lecz tym razem zaprojektować urządzenie, które pogodzi pozornie sprzeczne założenia.

Z jednej strony chciałem, by odznaczało się ono dużą prostotą implementacji oraz nieskomplikowaną obsługą, a z drugiej strony – efektownym i nowoczesnym interfejsem użytkownika. Nie ukrywam, że inspiracją do powstania niniejszego projektu był zakupiony przeze mnie na chińskim portalu sprzedażowym prosty zegar biurkowy, wyposażony w bardzo efektowny, graficzny wyświetlacz VFD. Dodajmy: bardzo efektowny, ale niewielki, gdyż konstrukcja dużych wyświetlaczy tego typu, zwłaszcza graficznych, jest niezwykle kosztowna i w zasadzie odchodzi do lamusa. Zaintrygowany wspomnianym rozwiązaniem postanowiłem skonstruować urządzenie o zbliżonej funkcjonalności, lecz wyposażone w znacznie większy wyświetlacz graficzny, a ponieważ z założenia urządzenie miało być proste w implementacji i niedrogie w konstrukcji – do roli elementów interfejsu użytkownika wybrałem popularne, dość duże matryce LED o organizacji 5×7 pikseli. Przyznam szczerze, że przez chwilę zastanawiałem się nad zastosowaniem programowalnych diod LED, ale biorąc pod uwagę niezbędną liczbę takich elementów (a co za tym idzie – koszt ich zakupu), jak i trudność późniejszej implementacji zrezygnowałem z tego pomysłu, pozostając przy wspomnianych już matrycach. I właśnie na bazie powyższych założeń powstał projekt urządzenia matrixClock, którego schemat pokazano na **rysunku 1**.

Jak widać, zaprojektowano bardzo prosty system mikroprocesorowy, którego serce stanowi niewielki, ale nowoczesny mikrokontroler ATtiny806 firmy Microchip (dawniej Atmel), taktowany wewnętrznym oscylatorem RC o częstotliwości 10 MHz i realizujący całą założoną funkcjonalność urządzenia. Mikrokontroler nasz steruje pracą grupy czterech 8-bitowych rejestrów przesuwnych (szeregowo-równoległych) typu STPIC6C595 (wyprowadzenia PB5…PB2 mikrokontrolera), dzięki którym realizuje obsługę 6 matrycowych wyświetlaczy LED w konfiguracji wspólnej anody (wyprowadzenia PA7…PA1 mikrokontrolera) oraz 5 dodatkowych diod LED, oznaczonych jako LED7…LED11. Ponadto obsługuje zegar czasu rzeczywistego z podtrzymaniem bateryjnym pod postacią układu MCP79410-I/SN firmy Microchip – oraz prostą klawiaturę złożoną z 4 przycisków typu microswitch (wyprowadzenia PC3…PC0 mikrokontrolera), przeznaczonych do obsługi urządzenia. W ramach ostatniej z wymienionych funkcjonalności układ – w celu eliminacji drgań styków oraz detekcji

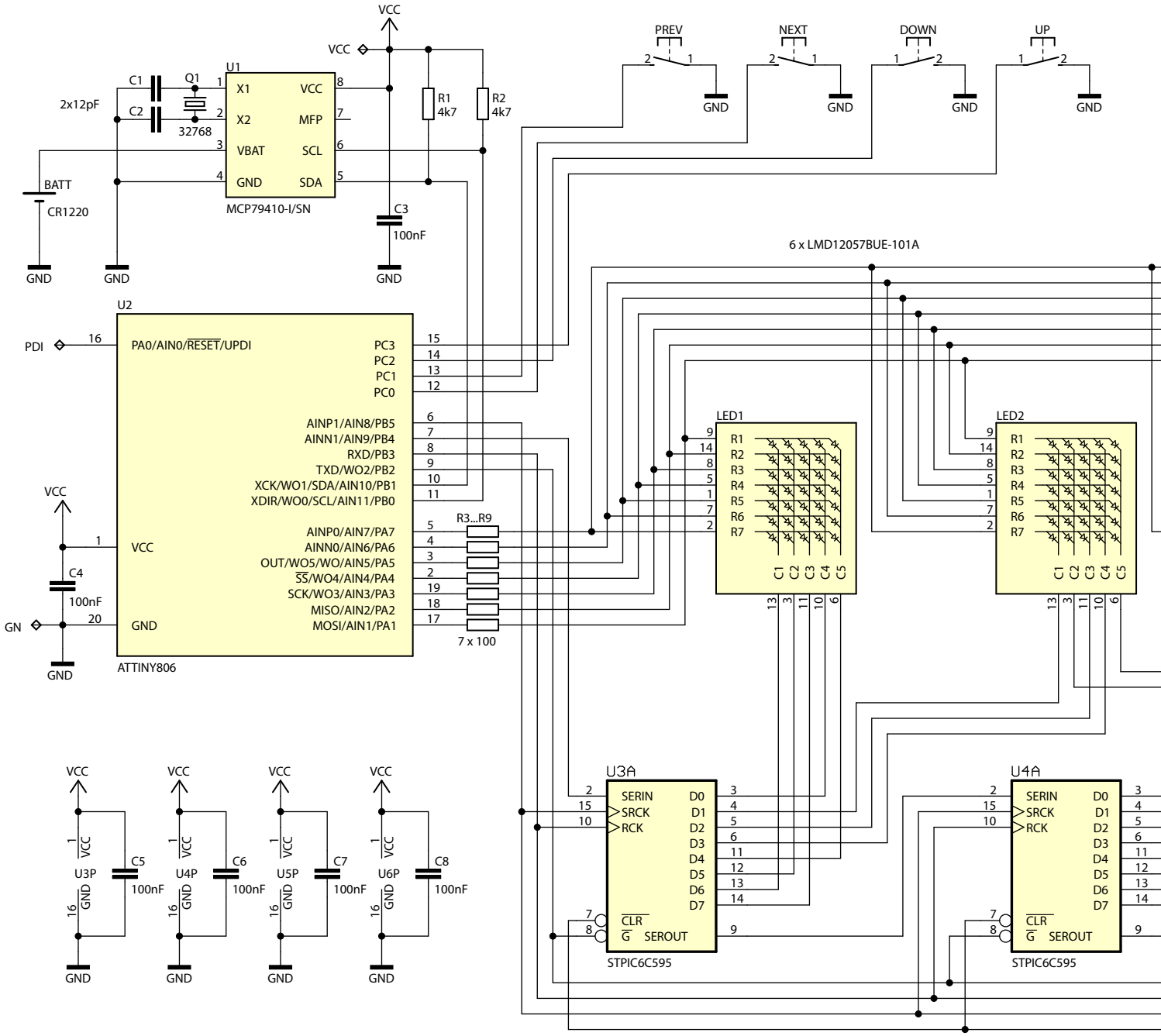
Wykaz elementów:

Rezystory: (obudowy SMD 0805)
R1, R2: 4.7 kΩ
R3...R9: 100 Ω
R10: 1 kΩ
R11...R15: 100 Ω

Kondensatory: (obudowy SMD 0805)
C1, C2: ceramiczny X7R 12 pF
C3...C10: ceramiczny X7R 100 nF
C11: tantalowy 100 µF/10V (obudowa C/6032-28R)

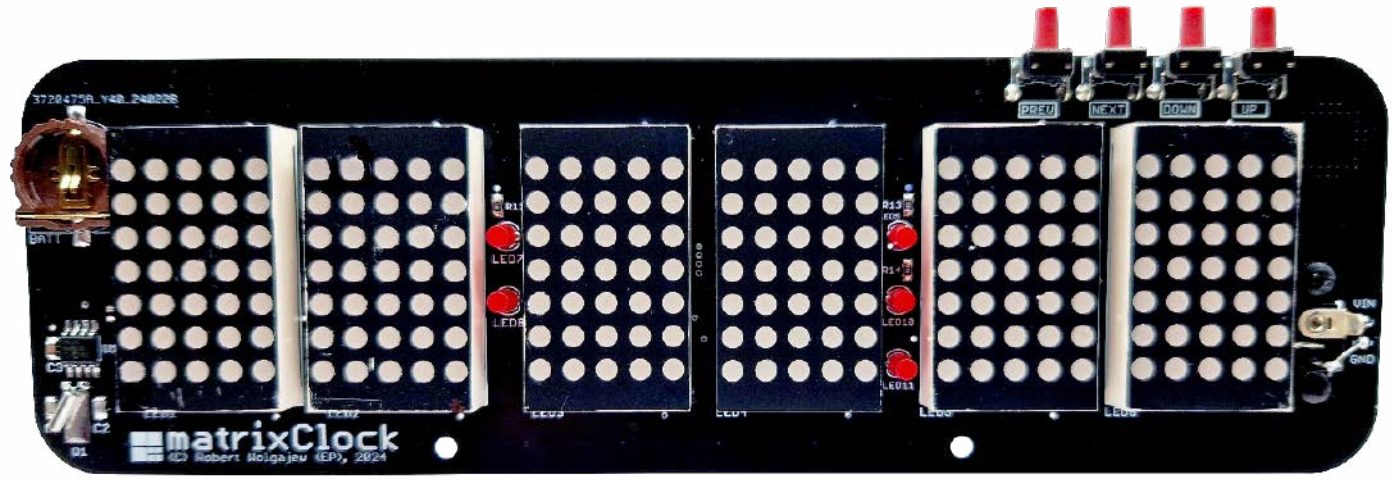
Półprzewodniki:
U1: MCP79410-I/SN (obudowa SO08)
U2: ATTINY806 (obudowa SO20)
U3...U6: STPIC6C595 (obudowa SO16)
U7: 78M05 (obudowa DPAK)
LED1...LED6: wyświetlacz matrycowy LMD12057BUE-101A, OSK351541-BR lub podobny o wybranym kolorze
LED7...LED11: dioda LED ∅ 3 mm czerwona, płaska

Pozostałe:
Q1: rezonator kwarcowy zegarkowy 32768 Hz
BATT: gniazdo baterii CR1220 typu CONNFLY DS1092-12-N8S
PREV, NEXT, DOWN, UP: microswitch TACT kątowy do montażu przewlekanego typu TL1105SF250Q lub podobny (wysokość 6 mm)
JACK: gniazdo zasilające do montażu przewlekanego typu NEB/J 25 (Lumberg) lub podobne
CR1220: bateria litowa pastylkowa typu CR1220



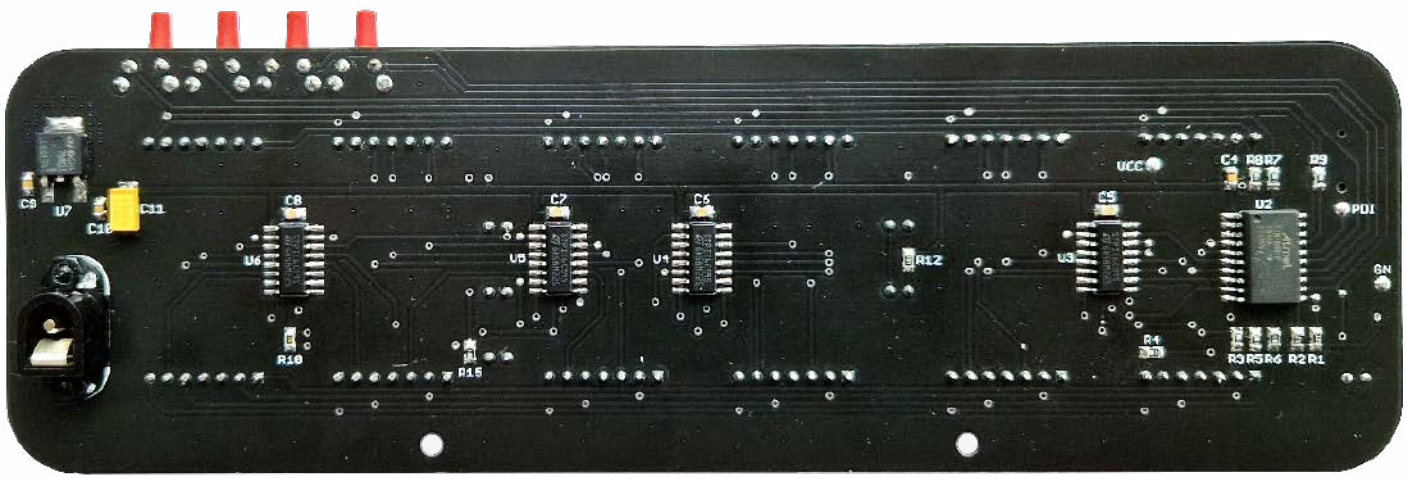
Rysunek 1. Schemat ideowy urządzenia matrixClock

krótkiego i długiego naciśnięcia każdego z przycisków – używa wbudowanego weń 16-bitowego układu czasowo-licznikowego TCB0 pracującego w trybie Periodic Interrupt. Uważnego Czytelnika zastanowi zapewne fakt wyboru nietypowych rejestrów przesuwnych zamiast zwyczajowych 74HC595. Jak się zapewne domyślacie, do obsługi tylu wyświetlaczy LED skorzystano ze znanego mechanizmu multipleksowania, a że do wysterowania mamy aż 32 wspólne katody (wyświetlaczy matrycowych LED1…LED6 i diod LED7…LED11), konieczne stało



Fotografia 1. Zmontowane urządzenie od strony warstwy TOP

się sterowanie tymi elementami dość dużym prądem, by wynikowa ich jasność była na akceptowalnym poziomie. W takim wypadku zastosowanie zwykłych 74HC595 okazałoby się niewystarczające, w związku z czym sięgnięto po element, na którego równoległych wyjściach zintegrowano tranzystory mocy DMOS pozwalające na przepływ prądu rzędu 100 mA na wyjście (przy aktywnych wszystkich wyjściach!). Dość egzotyczne może się również wydawać przyporządkowanie poszczególnych wyjść rejestrów przesuwnych do wspólnych



Fotografia 2. Zmontowane urządzenie od strony warstwy BOTTOM

katod elementów LED, gdyż pozornie nie ma w nim większego sensu. W sensie elektrycznym rzeczywiście tak można to postrzegać, lecz przyporządkowanie, o którym mowa powyżej, wynika z chęci uproszczenia projektu obwodu drukowanego, zaś wszelkie niedogodności z niego wynikające zostaną zniwelowane na drodze programowej.

Zegar czasu rzeczywistego jest obsługiwany przy użyciu interfejsu TWI, będącego funkcjonalnym odpowiednikiem standardu I²C firmy Philips. Już teraz zwrócę uwagę, że jeśli chcemy, by 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.

Wyświetlacze LED i diody LED7...LED11 są sterowane w trybie multipleksowym z częstotliwością odświeżania 60 Hz. Plik nagłówkowy zawierający najważniejsze definicje używane do obsługi multipleksu pokazano na **listingu 1**, zaś na **listingu 2** można zobaczyć stałe definiujące wzorce poszczególnych znaków. Zastosowane rozwiązania programowe, mające na celu optymalizację szybkości wykonywania kodu, objaśniono już przy okazji projektu miernika pojemności cMeter, opublikowanego w EP 05/2024 – zainteresowanych Czytelników zachęcamy do zapoznania się z zamieszczonym tam szczegółowym opisem. Funkcję konfigurującą mechanizm multipleksowania i dokonującą niezbędnych ustawień sprzętowych pokazano na **listingu 3**. Dalej, na **listingu 4** zamieszczono funkcję obsługi

```
//Definicje dla portów sterujących rejestrem przesuwnym
#define SER_REG_PORT_NAME PORTB
#define SER_REG_SRCK_MASK PIN5_bm
#define SER_REG_RCK_MASK PIN3_bm
#define SER_REG_SEROUT_MASK PIN4_bm
#define SER_REG_G_MASK PIN2_bm

#define SER_REG_AS_OUTPUTS SER_REG_PORT_NAME.DIRSET = SER_REG_SRCK_MASK|SER_REG_RCK_MASK|SER_REG_SEROUT_MASK|SER_REG_G_MASK

#define SER_REG_SRCK_SET SER_REG_PORT_NAME.OUTSET = SER_REG_SRCK_MASK
#define SER_REG_SRCK_RESET SER_REG_PORT_NAME.OUTCLR = SER_REG_SRCK_MASK
#define SER_REG_SRCK_TICK SER_REG_SRCK_SET; SER_REG_SRCK_RESET

#define SER_REG_RCK_SET SER_REG_PORT_NAME.OUTSET = SER_REG_RCK_MASK
#define SER_REG_RCK_RESET SER_REG_PORT_NAME.OUTCLR = SER_REG_RCK_MASK
#define SER_REG_RCK_TICK SER_REG_RCK_SET; SER_REG_RCK_RESET

#define SER_REG_SEROUT_SET SER_REG_PORT_NAME.OUTSET = SER_REG_SEROUT_MASK
#define SER_REG_SEROUT_RESET SER_REG_PORT_NAME.OUTCLR = SER_REG_SEROUT_MASK

#define SER_REG_G_SET SER_REG_PORT_NAME.OUTSET = SER_REG_G_MASK
#define SER_REG_G_RESET SER_REG_PORT_NAME.OUTCLR = SER_REG_G_MASK

//Definicje dla portu wspólnych anod wyświetlacza LED
#define COM_ANODE_PORT_NAME PORTA
#define COM_ANODE_AS_OUTPUTS COM_ANODE_PORT_NAME.DIRSET = 0xFF

//Definicje dla funkcji wyświetlających
#define NORMAL 0
#define BLINKING 1

//Deklaracje zmiennych globalnych
extern uint8_t Cols[32];                        //Zmienna przechowująca zawartość wyświetlacza LED (30 kolumn liczonych od lewej do prawej
                                                //+ 2 dodatkowe upraszczające kod ISR)
extern volatile uint8_t Colon, Dot, Blinking;   //Zmienne przechowujące stan dwukropków, kropki i migania
extern volatile uint8_t readyForUpdate;         //Zezwolenie na atomową zmianę zmiennych

void initMultiplex(void);
void showChar(uint8_t Char, uint8_t Position, uint8_t Offset, uint8_t Blink);
```

Listing 1. Plik nagłówkowy mechanizmu multipleksowania

```
//Tablica przechowująca wzorce znaków
const uint8_t Font5x8[] PROGMEM =
{
    0x00, 0x00, 0x00, 0x00, 0x00,  //spacja
    0x00, 0x00, 0x5F, 0x00, 0x00,  // !
    0x00, 0x07, 0x00, 0x07, 0x00,  // “
    0x14, 0x7F, 0x14, 0x7F, 0x14,  // #
    0x24, 0x2A, 0x7F, 0x2A, 0x12,  // $
    0x23, 0x13, 0x08, 0x64, 0x62,  // %
    0x36, 0x49, 0x55, 0x22, 0x50,  // &
    0x00, 0x05, 0x03, 0x00, 0x00,  // ‘
    0x00, 0x1C, 0x22, 0x41, 0x00,  // (
    0x00, 0x41, 0x22, 0x1C, 0x00,  // )
    0x08, 0x2A, 0x1C, 0x2A, 0x08,  // *
    0x08, 0x08, 0x3E, 0x08, 0x08,  // +
    0x00, 0x50, 0x30, 0x00, 0x00,  // ,
    0x08, 0x08, 0x08, 0x08, 0x08,  // -
    0x00, 0x30, 0x30, 0x00, 0x00,  // .
    0x20, 0x10, 0x08, 0x04, 0x02,  // /
    0x3E, 0x51, 0x49, 0x45, 0x3E,  // 0
    0x00, 0x42, 0x7F, 0x40, 0x00,  // 1
    0x42, 0x61, 0x51, 0x49, 0x46,  // 2
    0x21, 0x41, 0x45, 0x4B, 0x31,  // 3
    0x18, 0x14, 0x12, 0x7F, 0x10,  // 4
    0x27, 0x45, 0x45, 0x45, 0x39,  // 5
    0x3C, 0x4A, 0x49, 0x49, 0x30,  // 6
    0x01, 0x71, 0x09, 0x05, 0x03,  // 7
    0x36, 0x49, 0x49, 0x49, 0x36,  // 8
    0x06, 0x49, 0x49, 0x29, 0x1E,  // 9
    0x3E, 0x51, 0x49, 0x45, 0x3E,  // : -> 0 - ponownie, dla mechanizmu animacji
};

//Tablica przechowująca kolejność (od prawej do lewej) poszczególnych kolumn wyświetlacza
//LED widzianą od strony rejestrów przesuwnych. 1->DOT, 0->COLON
const uint8_t colPattern[32] = {25, 26, 24, 31, 27, 30, 18, 29, 28, 19, 17, 22, 16, 23, 21, 10, 20, 9, 8, 11, 15, 12, 14, 13, 2, 7, 3, 6, 5,
4, 1, 0};
```

Listing 2. Definicje niezbędnych stałych mechanizmu multipleksowania

Ustawienia Fuse-bitów:
FREQSEL[1:0]: 10[1]
RSTPINCFG[1:0]: 00[2]
SUT[2:0]: 111[1]
EESAVE: 0[1]

[1] ustawienie domyślne producenta
[2] szczegóły w treści artykułu

```
void initMultiplex(void)
{
    //Porty sterujące rejestrami przesuwnymi jako wyjściowe ze stanami nieaktywnymi (0)
    SER_REG_AS_OUTPUTS;
    //Port wspólnych anod, jako wyjściowy ze stanami nieaktywnymi (0)
    COM_ANODE_AS_OUTPUTS;
    //Konfiguracja Timera TCA0 odpowiedzialnego za mechanizm multipleksowania. Stosowne przerwanie
    //systemowe wywoływane 1920 razy na sekundę, czyli 60 razy na sekundę dla każdej kolumny LED/diod dwukropka i kropki
    TCA0.SINGLE.PER = 5207;
    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1_gc|TCA_SINGLE_ENABLE_bm;
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
}
```

Listing 3. Funkcja konfigurująca mechanizm multipleksowania

przerwania od przepełnienia licznika TCA0, odpowiedzialną za realizację mechanizmu multipleksowania wyświetlacza LED.

Listing 5 pokazuje funkcję wyświetlającą wzorzec znaku na elemencie LED. Jak widać, funkcja przyjmuje zarówno argument przesunięcia w pionie wzorca znaku o liczbę zdefiniowanych pikseli obrazu (w zakresie 0…8), co zostanie użyte w mechanizmie animacji zmian wyświetlanych cyfr – jak i argument odpowiedzialny za miganie wyświetlanej treści, co znajdzie z kolei zastosowanie w menu ustawień urządzenia (do wyróżnienia wartości poddawanej edycji). To już wszystko, jeśli chodzi o funkcje obsługi wyświetlacza LED – przejdźmy zatem do grupy funkcji odpowiedzialnych za obsługę interfejsu TWI. Nie jest to szczególnie nowatorskie rozwiązanie, jednak interfejs TWI w nowych mikrokontrolerach AVR Tiny z serii 0 różni się znacząco – pod względem sposobu obsługi, rozmieszczenia i znaczenia rejestrów konfiguracyjnych – od starszych wersji tych mikrokontrolerów, dlatego warto mu się przyjrzeć bliżej. Warto podkreślić, że prezentowane rozwiązanie nie będzie na wskroś uniwersalne (zależało mi bowiem na prostocie rozwiązań programistycznych), lecz w tak prostych zastosowaniach, jak omówione tutaj, moim zdaniem okazuje się wystarczająco dobre. Stosowny driver (w tym używający przerwań TWI) możemy zresztą wygenerować automatycznie z poziomu środowiska Microchip Studio, lecz jak możecie się sami przekonać, rozwiązanie proponowane przez producenta jest bardzo skomplikowane. Dzieje się tak, gdyż – jeśli ma pozostać w pełni uniwersalne – musi operować na dość dużym poziomie abstrakcji.

Zacznijmy od pliku nagłówkowego, którego ciało pokazano na **listingu 6**. Dalej, na **listingu 7**, zaprezentowano ciało funkcji inicjalizującej interfejs TWI mikrokontrolera AVR Tiny 0-series. Na **listingu 8** pokazano z kolei ciało funkcji odpowiedzialnej za wygenerowanie sygnału START interfejsu I²C i przesłanie adresu slave'a (wraz z bitem kierunku R/W), zaś na **listingu 9** – ciało funkcji odpowiedzialnej za wygenerowanie sygnału STOP interfejsu I²C. I na koniec dwie kluczowe funkcje, pozwalające na zapis i odczyt bajtu poprzez interfejs I²C, których ciała pokazano odpowiednio na **listingach 10** i **11**.

Skoro wiemy już, jak obsługiwać interfejs TWI naszego mikrokontrolera, pora na omówienie modułu obsługi zegara MCP79410-I/SN.

```
ISR(TCA0_OVF_vect)
{
    static uint8_t Idx, bigTick;
    static uint16_t smallTick;

    //Kasujemy flagę OVF, gdyż nie jest kasowana sprzętowo
    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;

    //Wyłączamy wszystkie bufory wyjściowe rejestrów przesuwnych dezaktywując tym samym kolumny wyświetlaczy LED
    SER_REG_G_SET;

    //Obsługujemy Tick migania - co 0.25 s
    if(++smallTick == 480)
    {
        smallTick = 0;
        bigTick ^= 1;
    }

    //Wybieramy kolejną z rzędu kolumnę wyświetlaczy LED wysyłając stosowny ciąg danych do rejestrów przesuwnych
    //Zaczynamy od kolumn wyświetlaczy LED po czym przechodzimy do kolumn sterujących dwukropkami i kropką
    for(uint8_t i=0; i<32; ++i)
    {
        if(i == colPattern[Idx])
        {
            if(Idx < 30)                                                    //Kolumny wyświetlaczy LED
            {
                if(Blinking & (1 << (Idx/5)))
                {
                    if(bigTick) SER_REG_SEROUT_SET; else SER_REG_SEROUT_RESET;
                }
                else SER_REG_SEROUT_SET;
            }
            else if(Idx == 30)                                              //30 -> DOT (kropka)
            {
                if(Dot) SER_REG_SEROUT_SET; else SER_REG_SEROUT_RESET;
            }

            else                                                            //31 -> COLON (dwukropki)
            {
                if(Colon) SER_REG_SEROUT_SET; else SER_REG_SEROUT_RESET;
            }
        }
        else SER_REG_SEROUT_RESET;

        //Sygnał zegarowy
        SER_REG_SRCK_TICK;
    }
    //Transfer stanów rejestrów przesuwnych do ich buforów wyjściowych (jeszcze nieaktywnych)
    SER_REG_RCK_TICK;
    //Na port wyjściowy wspólnych anod wystawiamy wzór do wyświetlenia
    COM_ANODE_PORT_NAME.OUT = Cols[Idx];
    //Włączamy wszystkie bufory wyjściowe rejestrów przesuwnych aktywując tym samym kolumny wyświetlaczy LED
    SER_REG_G_RESET;

    //Wybieramy kolejną z rzędu kolumnę. Opcjonalnie zezwolenie na atomową zmianę zmienej Cols[] w funkcji Main
    Idx = (Idx +1) & 0x1F;
    if(Idx == 0) readyForUpdate = 1; else readyForUpdate = 0;
}
```

Listing 4. Funkcja obsługi przerwania realizująca mechanizm multipleksowania

```
void showChar(uint8_t Char, uint8_t Position, uint8_t Offset, uint8_t Blink)
{
    uint8_t prevByte, nextByte;
    uint16_t Index;

    //Ustalamy index początku wzorca znaku, który to zamierzamy wyświetlić (odejmujemy 32, gdyż tablica zaczyna się od spacji)
    Index = (Char - ' ') * 5;

    //Czekamy na zezwolenie na aktualizację zawartości wyświetlacza LED
    readyForUpdate = 0;
    while(readyForUpdate == 0);

    //Aktualizujemy zawartość wyświetlacza uwzględniając przesunięcie wzorca cyfry
    for(uint8_t i=0; i<5; ++i)
    {
        prevByte = (uint8_t) pgm_read_byte(&Font5x8[Index]) >> Offset;
        nextByte = (uint8_t) pgm_read_byte(&Font5x8[Index+5]) << (8-Offset);

        Cols[i+(Position*5)] = prevByte|nextByte;
        Index++;
    }
    //Aktualizujemy stan zmiennej odpowiadającej za miganie
    if(Blink == BLINKING) Blinking |= (1<<Position); else Blinking &= ~(1<<Position);
}
```

Listing 5. Funkcja odpowiedzialna za wyświetlenie wzorca znaku na wyświetlaczu LED

Tradycyjnie zacznijmy od pliku nagłówkowego, którego ciało pokazano na **listingu 12**.

Jak widać, wprowadzono dwa dodatkowe typy strukturalne (*timeType*, *dateType*), odpowiedzialne za przechowywanie oraz przetwarzanie czasu i daty wbudowanego zegara RTC. Następnie, na **listingu 13**, pokazano funkcję, której zadaniem jest konfiguracja cech sprzętowych zegara czasu rzeczywistego. Argument wywołania tejże funkcji decyduje o ustawieniach RTC (np. ustawieniach wyjścia MFP) i w przypadku naszego urządzenia przyjmuje wartość:

```
MFP_AS_SQUARE_OUTPUT|NO_ALARMS_ACTIVE|INTERNAL_
OSCILLATOR|SQUARE_1HZ;
```

Na **listingu 14** pokazano z kolei dwie proste funkcje narzędziowe, umożliwiające odczyt i zapis czasu RTC, zaś na **listingu 15** – bliźniacze dwie funkcje odczytu i zapisu, ale tym razem daty.

Omówiliśmy kwestie implementacyjne, w związku z czym możemy przejść do schematu montażowego naszego urządzenia, który to pokazano na **rysunku 2**.

Na potrzeby urządzenia zaprojektowano kompaktową, dwustronną płytkę drukowaną ze zdecydowaną przewagą elementów SMD lutowanych po obu stronach laminatu. Montaż urządzenia rozpoczynamy od warstwy BOTTOM, gdzie w pierwszej kolejności przylutowujemy wszystkie półprzewodniki. Proces ten najłatwiej wykonać przy użyciu stacji lutowniczej na gorące powietrze (tzw. hot-air) i odpowiednich stopów lutowniczych. Jeśli jednak nie dysponujemy tego rodzaju sprzętem, można również zastosować metodę „zastępczą" z użyciem typowej stacji kolbowej i plecionki rozlutowniczej. Następnie lutujemy elementy bierne, po czym przechodzimy na warstwę TOP. Tutaj, inaczej niż poprzednio, w pierwszej kolejności przylutowujemy wszystkie elementy bierne, następnie montujemy rezonator kwarcowy, układ RTC (U1), gniazdo baterii CR1220 (BATT) a na samym końcu elementy LED (wyświetlacze matrycowe i diody LED → w jednej płaszczyźnie) oraz przyciski PREV, NEXT, DOWN i UP. W tym momencie wracamy na warstwę BOTTOM i montujemy (wsuwając w dedykowany otwór) gniazdo zasilające

```
//Definicje portów magistrali I²C
#define I²C_PORT_NAME PORTB
#define I²C_SDA_PINCTRL_REG PIN1CTRL                                    //PB1
#define I²C_SCL_PINCTRL_REG PIN0CTRL                                    //PB0

//Definicje częstotliwości magistrali oraz czasu narastania zboczy
#define I²C_FREQUENCY 400000UL                                          //Częstotliwość magistrali [Hz]
#define I²C_RISE_TIME 100UL                                             //Czas narastania zboczy zależny od impedancji magistrali [ns]
#define I²C_BAUD (uint8_t)((((((float) F_CPU/(float) I²C_FREQUENCY)) - 10 - ((float) F_CPU * I²C_RISE_TIME/1000000))) / 2)

//Definicje bitu potwierdzenia (ACK)
#define NACK 0
#define ACK 1

//Definicje statusów wykonania funkcji
#define OK 0
#define FAILED 255
```

Listing 6. Plik nagłówkowy modułu obsługi interfejsu I²C mikrokontrolerów AVR Tiny 0-series

```
void i2cInit(void)
{
    //Podciągnięcie portów SDA i SCL pod VCC
    I²C_PORT_NAME.I²C_SDA_PINCTRL_REG = PORT_PULLUPEN_bm;
    I²C_PORT_NAME.I²C_SCL_PINCTRL_REG = PORT_PULLUPEN_bm;

    TWI0.MBAUD = I²C_BAUD;                                              //Ustawienie częstotliwości magistrali
    TWI0.MCTRLA = TWI_ENABLE_bm;                                        //Włączenie modułu I²C -> Tryb Master bez przerwań (tzw. pooling)
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;                                //Domyślny tryb magistrali (Idle)
}
```

Listing 7. Ciało funkcji inicjalizującej sprzęg TWI mikrokontrolera AVR Tiny 0-series

```
uint8_t i2cStart(uint8_t Address)
{
    //Wysyłamy sygnał START z adresem klienta. Adres zawiera w sobie bit R/W (bit 0) decydujący o kierunku transmisji
    TWI0.MADDR = Address;

    while (!(TWI0.MSTATUS & (TWI_WIF_bm | TWI_RIF_bm)));               //Czekamy na zakończenie operacji

    //Jeśli doszło do błędu lub arbitrażu na magistrali i dostęp do niej został utracony zwracamy FAILED i czekamy na zwolnienie magistrali
    if(TWI0.MSTATUS & TWI_ARBLOST_bm)
    {
        while(!(TWI0.MSTATUS & TWI_BUSSTATE_IDLE_gc));                  //Czekamy na zwolnienie magistrali
        return FAILED;
    }
    //Jeśli klient nie potwierdził swojego adresu wysyłamy sygnał STOP i czekamy na zwolnienie magistrali
    else if(TWI0.MSTATUS & TWI_RXACK_bm)
    {
        TWI0.MCTRLB |= TWI_MCMD_STOP_gc;                                //Wysyłamy sygnał STOP
        while(!(TWI0.MSTATUS & TWI_BUSSTATE_IDLE_gc));                  //Czekamy na zwolnienie magistrali
        return FAILED;
    }

    return OK;
}
```

Listing 8. Ciało funkcji odpowiedzialnej za wygenerowanie sygnału START interfejsu I²C i przesłanie adresu klienta

```
void i2cStop(void)
{
    TWI0.MCTRLB |= TWI_MCMD_STOP_gc;                                    //Wysyłamy sygnał STOP
    while(!(TWI0.MSTATUS & TWI_BUSSTATE_IDLE_gc));                      //Czekamy na zwolnienie magistrali
}
```

Listing 9. Ciało funkcji odpowiedzialnej za wygenerowanie sygnału STOP interfejsu I²C

```
uint8_t i2cWriteByte(uint8_t Byte)
{
    TWI0.MDATA = Byte;
    TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc;                                //Wysyłamy bajt danych
    while(!(TWI0.MSTATUS & TWI_WIF_bm))                                 //Czekamy na zakończenie transmisji

    //Jeśli doszło do błędu lub arbitrażu na magistrali i dostęp do niej został utracony zwracamy FAILED
    if(TWI0.MSTATUS & (TWI_ARBLOST_bm | TWI_BUSERR_bm)) return FAILED;

    return!(TWI0.MSTATUS & TWI_RXACK_bm);                               //Zwracamy bit ACK
}
```

uListing 10. Ciało funkcji odpowiedzialnej za zapis bajtu poprzez interfejs I²C

```
uint8_t i2cReadByte(uint8_t Ack)
{
    uint8_t Byte;

    while(!(TWI0.MSTATUS & TWI_RIF_bm));                              //Czekamy na zakończenie odczytu
    Byte = TWI0.MDATA;
    //Wysyłamy sygnał ACK, gdy oczekujemy więcej danych (i wznawiamy transmisję) lub sygnał NACK
    if(Ack) TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc|TWI_ACKACT_ACK_gc; else TWI0.MCTRLB = TWI_ACKACT_NACK_gc;

    return Byte;
}
```

Listing 11. Ciało funkcji odpowiedzialnej za odczyt bajtu poprzez interfejs I²C

```
//Definicje typów strukturalnych do obsługi zegara RTC
typedef struct
{
    uint8_t Hour;                           //0...23
    uint8_t Minute;                         //0...59
    uint8_t Second;                         //0...59
}timeType;

typedef struct
{
    uint8_t weekDay;                        //1...7
    uint8_t Day;                            //1...31
    uint8_t Month;                          //1...12
    uint8_t Year;                           //0...99
}dateType;

//Prototypy funkcji modułu
void RTCinit(uint8_t Settings);
void RTCwriteTime(timeType *Time);
void RTCreadTime(timeType *Time);
void RTCwriteDate(dateType *Date);
void RTCreadDate(dateType *Date);

//Definicje adresów układu MCP79410 w trybie zapisu/odczytu
#define MCP79410_WRITE_ADDR 0xDE
#define MCP79410_READ_ADDR 0xDF

//Definicje najważniejszych rejestrów sterujących i ich właściwości
#define TIME_START_REG 0x00
    #define START_OSCILLATOR (1<<7)
    #define STOP_OSCILLATOR (0<<7)

#define DATE_START_REG 0x03
    #define VBAT_ENABLE (1<<3)
    #define VBAT_DISABLE (0<<3)

#define CONTROL_REG 0x07
    #define MFP_AS_OUTPUT_1 (1<<7)
    #define MFP_AS_OUTPUT_0 (0<<7)
    #define MFP_AS_SQUARE_OUTPUT (1<<6)
    #define MFP_AS_NORMAL_OUTPUT (0<<6)
    #define NO_ALARMS_ACTIVE (0<<4)
    #define ALARM0_ACTIVE (1<<4)
    #define ALARM1_ACTIVE (2<<4)
    #define BOTH_ALARMS_ACTIVE (3<<4)
    #define EXTERNAL_OSCILLATOR (1<<3)
    #define INTERNAL_OSCILLATOR (0<<3)
    #define SQUARE_1HZ 0x00
    #define SQUARE_4096HZ 0x01
    #define SQUARE_8192HZ 0x02
    #define SQUARE_32768HZ 0x03

#define RAM_START_REG 0x20
```

Listing 12. Plik nagłówkowy modułu obsługi układu MCP79410

JACK, unieruchamiając je za pomocą wkrętów M2. Zaciski gniazda podłączamy za pomocą dwóch, krótkich przewodów (zachowując stosowną polaryzację) do pól lutowniczych VIN i GND po stronie TOP na obwodzie drukowanym. Na tym etapie urządzenie gotowe jest do uruchomienia.

Jak można zauważyć, zastosowano czarną soldermaskę – i to nie bez powodu. Obwód drukowany urządzenia zamontowany będzie w efektownej obudowie wykonanej w technologii druku 3D, w której płytę czołową stanowić ma przydymiona pleksi przesłaniająca zarówno wyświetlacz, jak i pozostałe elementy zamontowane PCB. Zależało mi, aby maksymalnie ukryć „wnętrzności" urządzenia – stąd pomysł na tego typu rozwiązanie, które w praktyce okazało się skuteczne.

Omówiliśmy kwestie montażowe, przejdźmy zatem do tematu obsługi naszego urządzenia. Zgodnie z tym, o czym wspomniałem już wcześniej, w tym celu przewidziano 4 przyciski funkcyjne umownie oznaczone jako ◁, ▷, △, ▽ – lub inaczej PREV, NEXT, UP, DOWN. Jak łatwo się domyślić, przyciski PREV i NEXT służą głównie do poruszania się po menu urządzenia, w tym do wyboru ustawień poddawanych regulacji (w menu ustawień urządzenia), zaś przyciski UP i DOWN przeznaczone są z zasady do regulacji wartości tych ustawień. Niemniej jednak bieżąca funkcjonalność wszystkich przycisków zależy w dużej mierze od miejsca w systemie menu, w jakim znajduje się urządzenie. Ponadto – biorąc pod uwagę, iż rozpoznawane jest zarówno krótkie, jak i długie naciśnięcie każdego z nich – nie sposób skrótowo opisać tego zagadnienia. Z tego względu na **rysunku 3** pokazano diagram obrazujący sposób obsługi systemu matrixClock oraz dostępne opcje menu.

W ramach funkcjonalności zegara przewidziano dodatkowe menu nazwane THERMO, dzięki któremu możemy wyświetlić temperaturę mierzoną przez urządzenie. Jest to możliwe wyłącznie wtedy, gdy do wyprowadzenia UPDI mikrokontrolera (PA0) dołączono opcjonalny scalony termometr, przy czym – co dość unikalne w tego typu

```
void RTCinit(uint8_t Settings)
{
    i2cStart(MCP79410_WRITE_ADDR);                                    //Adres MCP79410 do zapisu
    i2cWriteByte(CONTROL_REG);                                        //Adres startowy rejestru ustawień zegara RTC
    i2cWriteByte(Settings);                                           //Ustawienia zegara RTC
    i2cStop();
}
```

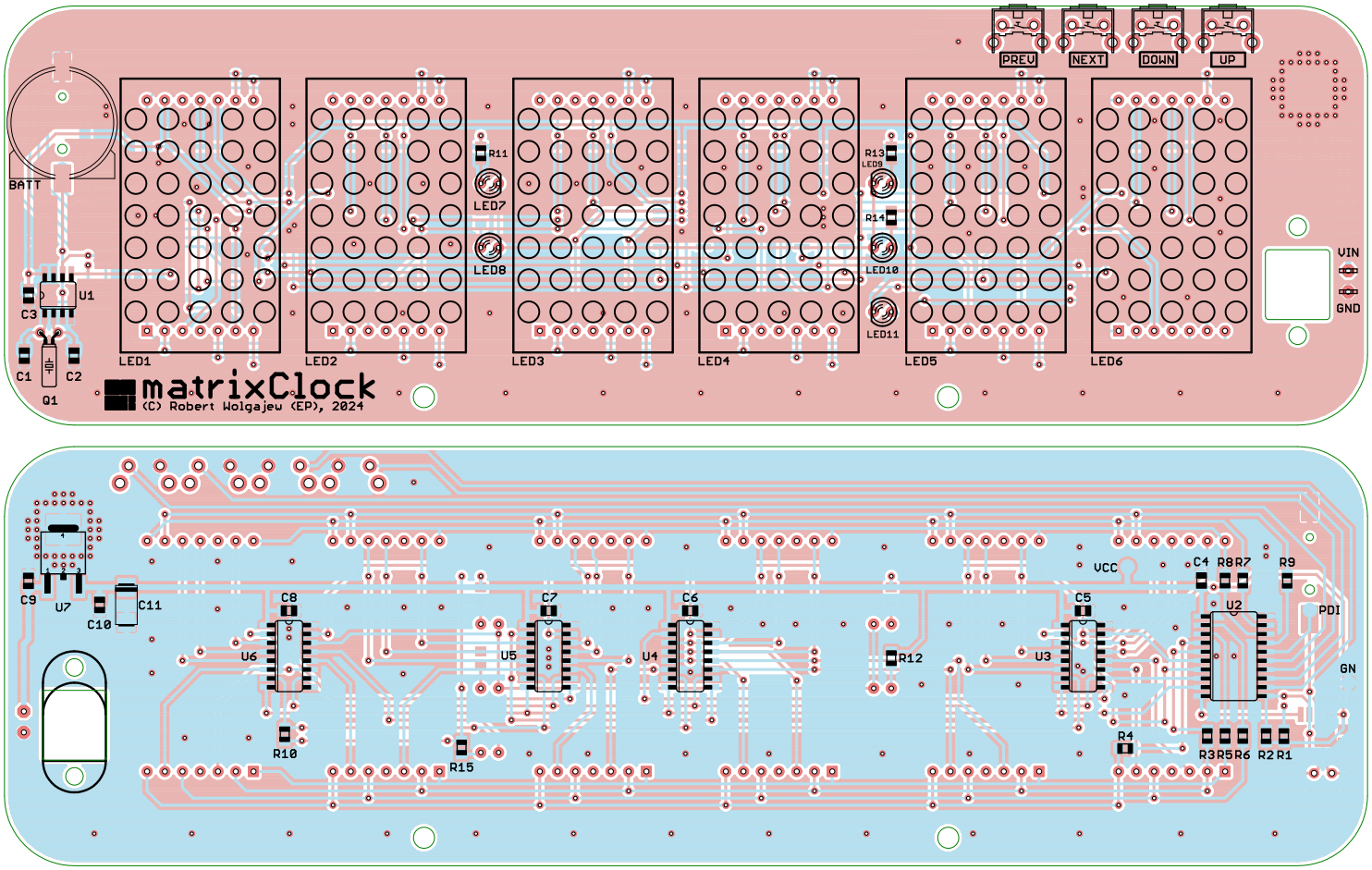
Listing 13. Funkcja odpowiedzialna za konfigurację układu MCP79410

```
void RTCwriteTime(timeType *Time)
{
    i2cStart(MCP79410_WRITE_ADDR);                                                  //Adres MCP79410 do zapisu
    i2cWriteByte(TIME_START_REG);                                                   //Adres startowy rejestru czasu (w tym przypadku sekund)
    i2cWriteByte(((Time->Second/10)<<4)|(Time->Second%10)|START_OSCILLATOR);        //Sekundy w zapisie BCD + start oscylatora
    i2cWriteByte(((Time->Minute/10)<<4)|(Time->Minute%10));                         //Minuty w zapisie BCD
    i2cWriteByte(((Time->Hour/10)<<4)|(Time->Hour%10));                             //Godziny w zapisie BCD (standardowo zapis 24-godzinny)
    i2cStop();
}

void RTCreadTime(timeType *Time)
{
    uint8_t readByte;

    i2cStart(MCP79410_WRITE_ADDR);                                                  //Adres MCP79410 do zapisu
    i2cWriteByte(TIME_START_REG);                                                   //Adres startowy rejestru czasu (w tym przypadku sekund)
    i2cStart(MCP79410_READ_ADDR);                                                   //Adres MCP79410 do odczytu

    readByte = i2cReadByte(ACK) & 0x7F;                                             //Sekundy w zapisie BCD - maskujemy bit pracującego
                                                                                    //oscylatora (bit7)
    Time->Second = ((readByte>>4)*10) + (readByte&0x0F);

    readByte = i2cReadByte(ACK);                                                    //Minuty w zapisie BCD
    Time->Minute = ((readByte>>4)*10) + (readByte&0x0F);

    readByte = i2cReadByte(NACK);                                                   //Godziny w zapisie BCD (standardowo zapis 24-godzinny)
    Time->Hour = (((readByte&0x30)>>4)*10) + (readByte&0x0F);
    i2cStop();
}
```

Listing 14. Funkcje przeznaczone do odczytu i zapisu czasu układu MCP79410

```
void RTCwriteDate(dateType *Date)
{
    i2cStart(MCP79410_WRITE_ADDR);                          //Adres MCP79410 do zapisu
    i2cWriteByte(DATE_START_REG);                           //Adres startowy rejestru daty (w tym przypadku dni tygodnia)
    i2cWriteByte(Date->weekDay|VBAT_ENABLE);                //Dzień tygodnia + aktywacja podtrzymania bateryjnego
    i2cWriteByte(((Date->Day/10)<<4)|(Date->Day%10));       //Dzień w zapisie BCD
    i2cWriteByte(((Date->Month/10)<<4)|(Date->Month%10));   //Miesiąc w zapisie BCD
    i2cWriteByte(((Date->Year/10)<<4)|(Date->Year%10));     //Rok w zapisie BCD
    i2cStop();
}

void RTCreadDate(dateType *Date)
{
    uint8_t readByte;

    i2cStart(MCP79410_WRITE_ADDR);                          //Adres MCP79410 do zapisu
    i2cWriteByte(DATE_START_REG);                           //Adres startowy rejestru daty (w tym przypadku dni tygodnia)
    i2cStart(MCP79410_READ_ADDR);                           //Adres MCP79410 do odczytu

    Date->weekDay = i2cReadByte(ACK) & 0x07;                //Dzień tygodnia

    readByte = i2cReadByte(ACK);                            //Dzień w zapisie BCD
    Date->Day = ((readByte>>4)*10) + (readByte&0x0F);

    readByte = i2cReadByte(ACK);                            //Miesiąc w zapisie BCD
    Date->Month = (((readByte & 0x10)>>4)*10) + (readByte&0x0F);

    readByte = i2cReadByte(NACK);                           //Rok w zapisie BCD
    Date->Year = ((readByte>>4)*10) + (readByte&0x0F);
    i2cStop();
}
```

Listing 15. Funkcje przeznaczone do odczytu i zapisu daty układu MCP79410



Rysunek 2. Schemat montażowy urządzenia: a) – warstwa TOP, b) – warstwa BOTTOM

układach – przewidziano możliwość obsługi dwóch, zupełnie różnych rodzajów sensorów, mianowicie:

- scalonego cyfrowego termometru typu DS18S20 (DS1820) wyposażonego w magistralę 1-wire,
- scalonego analogowego termometru typu LM35 z wyjściem napięciowym.

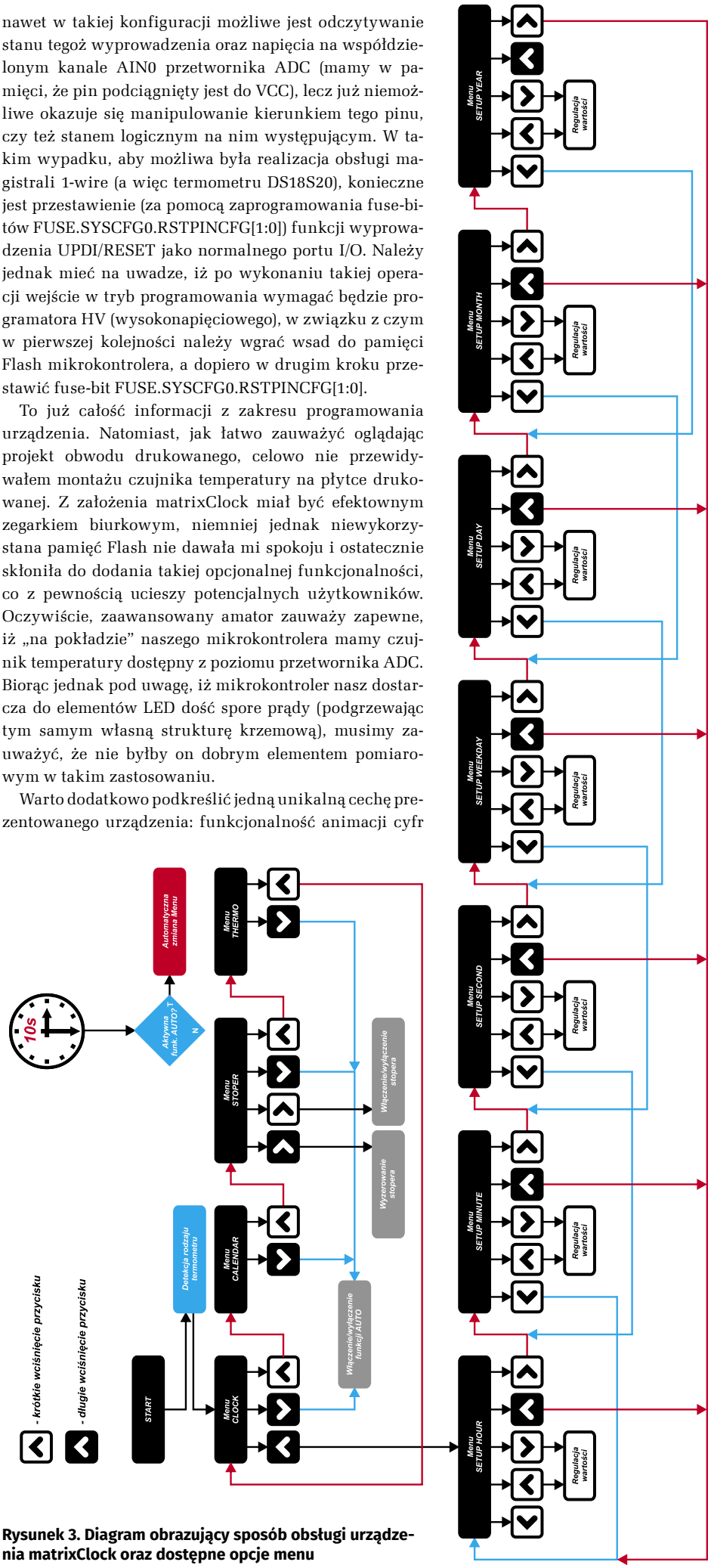
Detekcja rodzaju podłączonego termometru dokonywana jest jedynie podczas włączania urządzenia, a polega na wysłaniu sygnału RESET na magistralę 1-Wire (wyprowadzenie UPDI) i następującym po nim oczekiwaniu na sygnał PRESENCE wysyłany przez układ Slave (DS18S20). Przy braku odpowiedzi ze strony potencjalnie podłączonego układu Slave, urządzenie matrixClock zakłada, że podłączono analogowy czujnik temperatury, w związku z czym inicjalizuje przetwornik ADC wbudowany w strukturę mikrokontrolera, dzięki któremu w późniejszym czasie dokonuje pomiaru temperatury układu LM35. Oczywiście sposób podłączenia obu rodzajów czujników do portu UPDI mikrokontrolera jest odmienny i pokazano go na **rysunku 4**.

Co ważne, pod żadnym pozorem nie należy podłączać czujnika LM35 bez szeregowego rezystora o rezystancji 10 kΩ, gdyż układ – generując sygnał RESET magistrali 1-Wire – ściąga wyprowadzanie UPDI mikrokontrolera do masy i brak tego rezystora mógłby spowodować uszkodzenie zarówno czujnika LM35, jak i portu UPDI mikrokontrolera. Uważny Czytelnik dostrzeże w tym momencie pewną sprzeczność, jeśli chodzi o wybór wyprowadzenia UPDI jako realizującego obsługę opcjonalnych termometrów. I słusznie. Wyprowadzenie, o którym mowa, standardowo skonfigurowane jest do obsługi programowania mikrokontrolera (dzięki stosownemu ustawieniu fuse-bitów FUSE.SYSCFG0.RSTPINCFG[1:0]) i w takim wypadku nie nadaje się do realizacji jakiejkolwiek magistrali danych. Co prawda

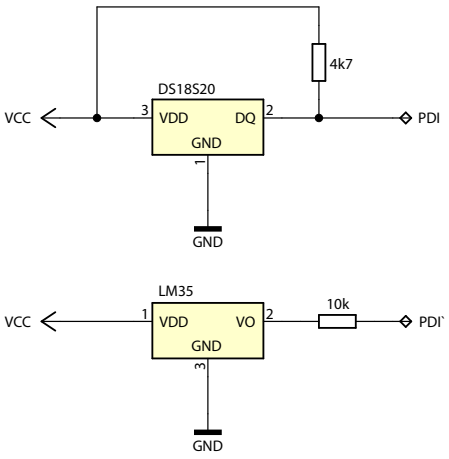
nawet w takiej konfiguracji możliwe jest odczytywanie stanu tegoż wyprowadzenia oraz napięcia na współdzielonym kanale AIN0 przetwornika ADC (mamy w pamięci, że pin podciągnięty jest do VCC), lecz już niemożliwe okazuje się manipulowanie kierunkiem tego pinu, czy też stanem logicznym na nim występującym. W takim wypadku, aby możliwa była realizacja obsługi magistrali 1-wire (a więc termometru DS18S20), konieczne jest przestawienie (za pomocą zaprogramowania fuse-bitów FUSE.SYSCFG0.RSTPINCFG[1:0]) funkcji wyprowadzenia UPDI/RESET jako normalnego portu I/O. Należy jednak mieć na uwadze, iż po wykonaniu takiej operacji wejście w tryb programowania wymagać będzie programatora HV (wysokonapięciowego), w związku z czym w pierwszej kolejności należy wgrać wsad do pamięci Flash mikrokontrolera, a dopiero w drugim kroku przestawić fuse-bit FUSE.SYSCFG0.RSTPINCFG[1:0].

To już całość informacji z zakresu programowania urządzenia. Natomiast, jak łatwo zauważyć oglądając projekt obwodu drukowanego, celowo nie przewidywałem montażu czujnika temperatury na płytce drukowanej. Z założenia matrixClock miał być efektownym zegarkiem biurkowym, niemniej jednak niewykorzystana pamięć Flash nie dawała mi spokoju i ostatecznie skłoniła do dodania takiej opcjonalnej funkcjonalności, co z pewnością ucieszy potencjalnych użytkowników. Oczywiście, zaawansowany amator zauważy zapewne, iż „na pokładzie" naszego mikrokontrolera mamy czujnik temperatury dostępny z poziomu przetwornika ADC. Biorąc jednak pod uwagę, iż mikrokontroler nasz dostarcza do elementów LED dość spore prądy (podgrzewając tym samym własną strukturę krzemową), musimy zauważyć, że nie byłby on dobrym elementem pomiarowym w takim zastosowaniu.

Warto dodatkowo podkreślić jedną unikalną cechę prezentowanego urządzenia: funkcjonalność animacji cyfr



Rysunek 3. Diagram obrazujący sposób obsługi urządzenia matrixClock oraz dostępne opcje menu



Rysunek 4. Sposób podłączenia obu rodzajów czujników temperatury do portu UPDI mikrokontrolera

wyświetlaczy, która dostępna jest w menu CLOCK i STOPER (zegara i stopera). We wspomnianych menu zmiana cyfr na wyświetlaczu matrycowym „okraszona" jest animacją, która przywodzi na myśl obracanie się bębna z cyframi, co miało miejsce w przypadku tradycyjnych, mechanicznych liczników w magnetofonach z lat 80. Takie rozwiązanie było również dostępne w dalekowschodnim zegarze z wyświetlaczem VFD, który posłużył za inspirację do powstania niniejszego projektu.

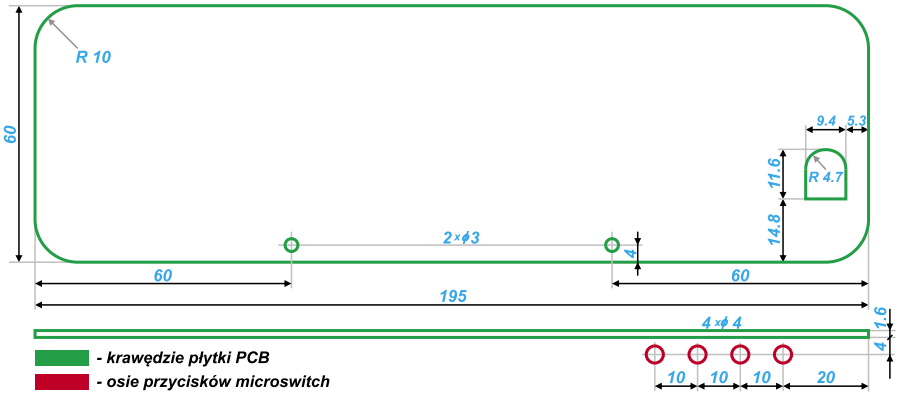
I na sam koniec zagadnień dotyczących obsługi naszego urządzenia – słowo komentarza na temat ostatniej funkcjonalności, w jakie je wyposażono, a mianowicie automatycznej zmiany menu urządzenia, którą włącza się i wyłącza poprzez długie wciśnięcie przycisku DOWN. Włączenie tej funkcji powoduje cykliczną zmianę (co 10 s) menu urządzenia w cyklu CLOCK → CALENDAR → STOPER → THERMO → CLOCK itd.

Obudowa

Na koniec naszego artykułu kilka niezbędnych słów uwagi należy się zagadnieniu projektu obudowy urządzenia matrixClock. Definiując stosowne założenia projektowe za cel postawiłem sobie przygotowanie estetycznej i nowoczesnej w swym kształcie obudowy, która w założeniach ma być ustawiana na biurku, przez co musi zostać wyposażona w niezbędne podpórki. Etapem wyjściowym do wykreowania poprawnego projektu było przygotowanie odpowiedniego obwodu drukowanego, który zarówno swym kształtem, jak i umieszczonymi nań elementami montażowymi upraszczałby późniejszy proces montażu. Kluczowe wymiary wspomnianego obwodu drukowanego urządzenia matrixClock pokazano na **rysunku 5**.

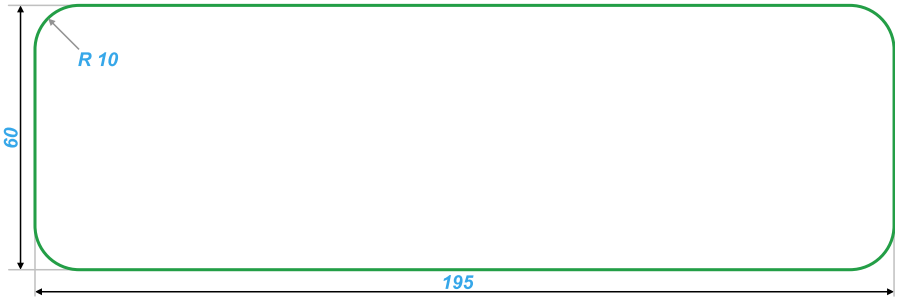
Bazując na powyższych założeniach przygotowano bardzo zwarty, 1-elementowy projekt obudowy urządzenia, którego rzut 3D pokazano na **rysunku 6**, zaś wybrany przekrój poprzeczny na **rysunku 7**.

Obwód drukowany urządzenia mocujemy do obudowy za pomocą 2 wkrętów.

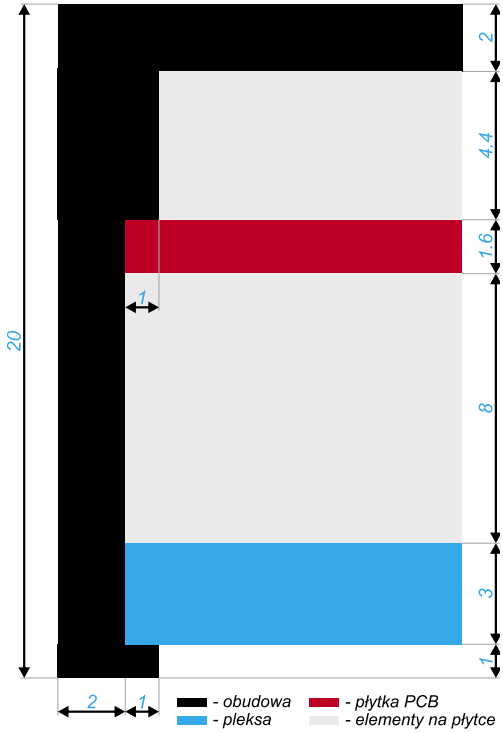


Rysunek 5. Kluczowe wymiary obwodu drukowanego urządzenia

Rysunek 6. Rzut 3D obudowy urządzenia



Rysunek 8. Wygląd płyty czołowej (pleksi) urządzenia



Rysunek 7. Przekrój poprzeczny obudowy urządzenia

Gniazdo zasilające (JACK) wystaje poza tylną płaszczyznę obudowy przez odpowiednio przygotowane wycięcie. Z uwagi na obecność mikroprzełączników (w górnej części obwodu drukowanego), które mają wystawać przez otwory ponad górną płaszczyznę obudowy, montaż PCB musimy wykonać z pewnym wyczuciem wsuwając go pod kątem do tak przygotowanej obudowy (by osie przełączników weszły w stosowne otwory). Następnie unieruchamiamy wsunięty obwód drukowany za pomocą 2 wkrętów, po czym montujemy półprzeźroczysty (przydymiony) kawałek pleksi stanowiący płytę czołową obudowy korzystając z dwóch niewielkich wystających rantów przytrzymujących go (pleksi musimy delikatnie wygiąć, by dało się ją umieścić w „świetle" obudowy). Zainteresowanym Czytelnikom podpowiem, iż sam zakupiłem taki kawałek pleksi na znanym portalu aukcyjnym (łącznie z usługą jego wycięcia) za przysłowiowe 5 zł. Aby zlecić taką usługę niezbędny będzie projekt (i stosowny plik) płyty czołowej urządzenia (pleksi), który pokazano na **rysunku 8**.

Na **fotografii 3** zaprezentowano natomiast wygląd obudowy wydrukowanej na niedrogiej, filamentowej drukarce 3D, zaś dla kontrastu, na zdjęciu tytułowym pokazano z kolei wygląd obudowy urządzenia wydrukowanej w technologii MJF (Multi Jet Fusion), dzięki usłudze jednej z dalekowschodnich firm produkujących obwody drukowane, o czym wspomniałem już w opisach projektów publikowanych wcześniej na łamach „Elektroniki Praktycznej". Szczerze polecam tego rodzaju rozwiązanie, gdyż – jak widać na **fotografii 4** – jakość takiego wydruku jest wprost fenomenalna.

Robert Wołgajew, EP

Fotografia 3. Widok obudowy urządzenia wydrukowanej w technologii FDM

Fotografia 4. Widok tylnej części obudowy urządzenia wydrukowanej w technologii MJF