Autor dziękuje Panu Jakubowi Cieślewiczowi z firmy Maritex za dostarczenie darmowych próbek diod LED typu APA-102-2C, które zostały użyte w aplikacji urządzenia.

Podstawowe parametry:
- zakres mierzonych temperatur: 10...34°C,
- rozdzielczość pomiaru temperatury: 1°C,
- dokładność pomiaru temperatury: ±0,5°C,
- dwa tryby wyświetlania:
- wyświetlenie linijki LED o płynnie zmieniającym się kolorze w miarę wzrostu temperatury (wyświetlenie paska diod LED podobnego do widma światła widzialnego),
- zapalenie się linijki diod LED w jednym kolorze, którego barwa zależeć będzie od mierzonej temperatury otoczenia,
- napięcie zasilania: 9...12 VDC,
- maksymalny prąd obciążenia: 0,25 A.

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

AVT5952	eT – wielokanałowy, bezprzewodowy system pomiaru temperatury (EP 9/2022)
AVT5949	Energooszczędny termometr LED (EP 8/2022)
AVT5892	Energooszczędny termometr z kalibracją (EP 10/2021)
AVT5635	Bezprzewodowy, energooszczędny system pomiaru temperatury (EP 8–9/2018)
AVT1999	2-kanałowy termometr MIN-MAX z alarmem (EP 8/2018)
AVT5623	4-kanałowy termometr z interfejsem Wi-Fi (EP 4/2018)
AVT5566	THPStation – rozbudowany termometr z Wi-Fi (EP 1/2017)
AVT5535	Termometr 2-kanałowy z interfejsem Bluetooth (EP 4/2016)
AVT5518	Termometr bezprzewodowy (EP 11/2015)
AVT1863	Termometr z interfejsem Bluetooth (EP 8/2015)
AVT1790	Termometr XXL (EP 2/2014)
AVT5489	8-kanałowy termometr z alarmem i wyświetlaczem LCD (EP 11/2013)
AVT5420	Wielopunktowy termometr z rejestracją (EP 10/2013)

* **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**.

Termometr RGB

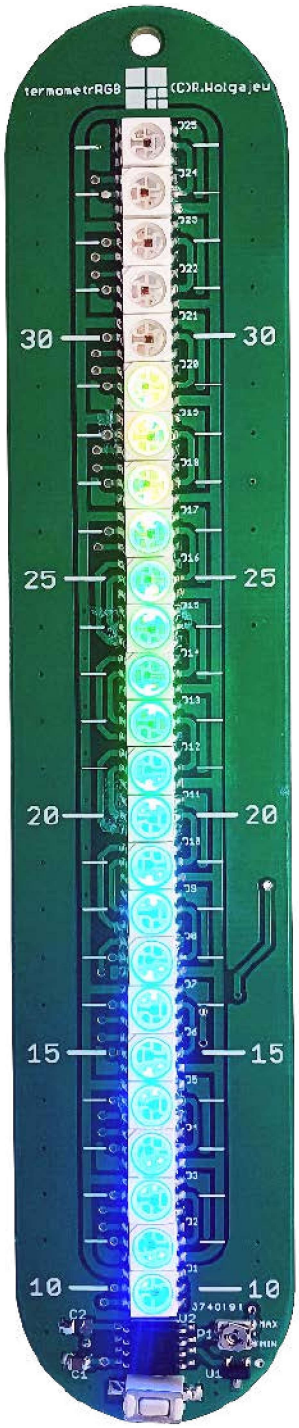
Decydując się na zaprojektowanie termometru, od razu odrzuciłem pokusę użycia jakiegokolwiek wyświetlacza jako elementu interfejsu użytkownika. Nawet na łamach naszego miesięcznika widziałem dziesiątki takich projektów, więc nie byłoby tu żadnego novum. Po krótkich przemyśleniach i rozwianiu kilku wątpliwości postanowiłem, że zaprojektuję termometr pokojowy wyposażony w analogową skalę temperatury złożoną z linijki diod LED RGB, które będą zastępowały rtęć (lub inne medium) tradycyjnego termometru tego typu.

Historia powstania tego projektu jest dość interesująca, choć zapewne nie nazbyt oryginalna. Otóż był taki czas, że szukałem jakiegoś ciekawego i zarazem praktycznego gadżetu elektronicznego, który mógłbym zastosować w domowym zaciszu. Dość szybko zdałem sobie sprawę z faktu, iż mimo wielu lat praktyki w projektowaniu systemów mikroprocesorowych w zasadzie nigdy nie wykonałem projektu termometru elektronicznego jako samodzielnego urządzenia. Zwykle była to funkcjonalność zaszyta w jakimś mniej lub bardziej skomplikowanym systemie docelowym. W sumie nie powinno to dziwić, wszak na rynku dostępnych jest mnóstwo urządzeń tego typu, które same w sobie nie są ani nazbyt oryginalne a już na pewno nieszczególnie skomplikowane. Pomyślałem, że nadszedł czas na mój autorski projekt o takiej ograniczonej funkcjonalności, a że lubię robić wszystko po swojemu zdecydowałem, iż zaprojektuję urządzenie inne, niż te dostępne w handlu.

Powiecie, iż nie ma w tym nic nowego, termometry z linijką LED to żadne odkrycie, ja jednak użyję diod LED typu RGB by zmieniającej się temperaturze towarzyszyła płynna zmiana koloru linijki LED i to w dwóch wersjach użytkowych. Pierwszy tryb pozwalał będzie na wyświetlenie linijki LED o płynnie zmieniającym się kolorze w miarę wzrostu temperatury generując wyświetlenie paska diod LED podobnego do widma światła widzialnego (nazwany trybem RGB), zaś drugi tryb pozwalał będzie na zapalenie się linijki diod LED w jednym kolorze, którego barwa zależeć będzie od mierzonej temperatury otoczenia (nazwany trybem COLOR).

Co więcej, projekt płytki drukowanej podporządkuję urządzeniu źródłowemu co oznacza, iż będzie on do złudzenia przypominał tradycyjny termometr pokojowy, przy czym zawężę zakres mierzonych temperatur do przedziału 10...34°C, co odpowiada zwyczajowym temperaturom, jakie notujemy w naszych mieszkaniach. Zawężenie to wynika wyłącznie z konieczności ograniczenia liczby diod LED (do 25 sztuk) tak, by wynikowy termometr nie był zbyt wysoki, gdyż, co już podkreślałem, ma imitować swój analogowy pierwowzór.

W tym miejscu stanąłem przed wyzwaniem wyboru odpowiednich elementów wykonawczych, a więc sterownika diod LED, jak i samych diod. Jak wiemy, aby płynnie sterować kolorem diody LED typu RGB należy zastosować 3 kanałowy sterownik PWM. Wynika



z tego, że skoro przewiduję zastosowanie 25 elementów tego typu to liczba niezbędnych kanałów wzrasta nam do 75. Trudno wyobrazić sobie mikrokontroler, który sprostałby tym wymaganiom a jeszcze trudniej wyobrazić sobie projekt płytki drukowanej o niewielkiej wielkości (wszak ma przypominać termometr pokojowy), na której upakowalibyśmy tyle odrębnych ścieżek. Co oczywiste, można byłoby zastosować jakiegoś rodzaju sterowanie matrycowe by ograniczyć liczbę koniecznych połączeń, ale biorąc pod uwagę liczbę wymaganych kanałów PWM i oczekiwaną rozdzielczość takiego sygnału (8-bitów) trudno mi sobie wyobrazić efektywne sterowanie bez użycia dość dużych prądów, by uzyskać wynikową jasność na akceptowalnym poziomie. Zresztą nawet w takim przypadku nie rozwiązujemy problemu ze skomplikowaniem rysunku obwodu drukowanego.

Pat? Otóż nie. Dość szybko zdałem sobie sprawę, iż jedynym sensownym rozwiązaniem tego rodzaju problemu konstrukcyjnego będzie zastosowanie adresowalnych diod LED RGB, których konstrukcja pozwala na uniknięcie wszystkich wspomnianych problemów. Diody takie, oprócz wyprowadzeń zasilających, wyposażone są w jakiś szeregowy interfejs komunikacyjny, przy użyciu którego dokonujemy ustawień koloru jej świecenia. Interfejs, o którym mowa, zaimplementowany jest w taki sposób (zarówno w kwestii sprzętowej, jak i logicznej), że diody takie połączone w łańcuchy mogą być indywidualnie adresowane, a co za tym idzie, każda z nich ma niezależne sterowanie. Sam przebieg PWM niezbędny do regulacji koloru jej świecenia generowany jest sprzętowo dzięki sterownikowi zabudowanemu na pokładzie takiego elementu.

Przejdźmy zatem do konkretów. Pierwszą myślą, jaka przyszła mi w tym czasie do głowy było zastosowanie bardzo popularnych i tanich elementów tego typu pod postacią diod z rodziny WS2811/WS2812 (i podobnych) lecz jednoprzewodowy interfejs komunikacyjny, w jaki je wyposażono wymaga dość sztywnych restrykcji czasowych (*timingów*) oraz, co wynika poniekąd z pierwszej właściwości, udostępnia dość ograniczone prędkości transmisji. Oczywiście w tak prostym zastosowaniu, jak termometr nie ma to praktycznie żadnego znaczenia, gdyż układ nasz nie robi nic więcej poza pomiarem i wyświetlaniem temperatury, więc można byłoby zastosować najprostszą implementację tego rodzaju medium transmisyjnego bazujące na funkcjach opóźniających (typu *_delay_us*), czyli blokującą działanie programu głównego na czas transmisji danych, lecz wiecie już, że…lubię zrobić wszystko po swojemu.

Diody LED RGB typu APA102

Dość szybko zorientowałem się, iż na rynku elementów półprzewodnikowych dostępne są diody LED o identycznej funkcjonalności, zaś pozbawione wad elementów rodziny WS2811/WS2812. Mowa o diodach typu APA102, które charakteryzują się następującymi właściwościami użytkowymi:

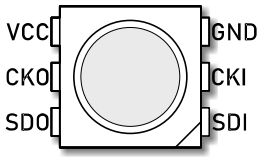
- 8-bitowa głębia składowych koloru (R, G, B),
- 5-bitowa regulacja jasności świecenia,
- 2-przewodowy, synchroniczny, szybki (15 MHz) interfejs komunikacyjny zgodny z SPI,
- duża jasność świecenia,
- dostępność gotowych łańcuchów diod LED tego rodzaju,
- niska cena (zwłaszcza na portalach aukcyjnych).

Jak widać, powyższe właściwości idealnie wpisują się w wymagania naszego systemu mikroprocesorowego, więc ich zastosowanie stało się naturalnym wyborem. Tak, jak wspomniano powyżej, dioda typu APA102 (w obudowie SMD PLCC-6 o wymiarach 5×5 mm) wyposażona jest w 6 wyprowadzeń:

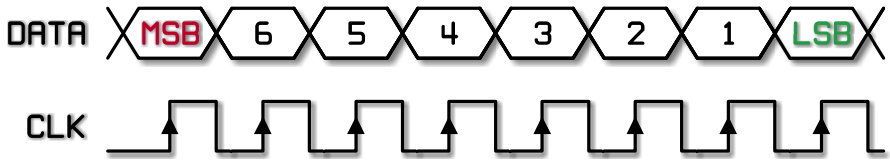
- wyprowadzenia zasilające: GND i VCC,
- wejście synchronicznego interfejsu komunikacyjnego (sygnał danych i zegar): SDI i CKI,
- wyjście synchronicznego interfejsu komunikacyjnego (sygnał danych i zegar): SDO i CKO.

Wygląd obudowy diody typu APA102 z zaznaczeniem nazw wyprowadzeń pokazano na **rysunku 1**. Jak zapewne się domyślacie, podobnie jak to ma miejsce w przypadku diod WS2811, diody typu APA102 łączy się w łańcuchy łącząc wyjścia interfejsu komunikacyjnego diody bieżącej z wejściami interfejsu komunikacyjnego diody kolejnej i tak dalej. Wejście interfejsu komunikacyjnego diody pierwszej, co oczywiste, łączy się z wyjściem tegoż interfejsu w mikrokontrolerze, zaś sama konstrukcja ramek danych i sposób działania sterownika „zabudowanego" w strukturze diody zapewnia odpowiednią synchronizację transmisji i niezbędne adresowanie.

Zacznijmy więc od podstaw. Na **rysunku 2** pokazano przebiegi sygnałów interfejsu komunikacyjnego diody APA102 (linia danych i zegar) w trakcie transmisji bajta danych. Jak widać, dane przesyłane są w kolejności od bitu najstarszego (MSB) do najmłodszego (LSB), zaś odczyt tych danych po stronie diody LED następuje na rosnącym zboczu sygnału zegarowego.



Rysunek 1. Wygląd obudowy diody typu APA102 z zaznaczeniem nazw wyprowadzeń



Rysunek 2. Przebiegi sygnałów interfejsu komunikacyjnego diody APA102 (linia danych i zegar) w trakcie transmisji bajta danych

Rysunek 3. Kompletna ramka transmisji diody APA102

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

Rezystory: (SMD0805)
R1: 36 kΩ 1%
R2: 35,7 kΩ 1%
R3: 10,5 kΩ 1%
R4: 2 kΩ 1%

Kondensatory:
C1, C2, C4, C6: 100 nF ceramiczny X7R (SMD0805)
C3: 100 µF/16 V elektrolityczny (PANASONIC_E)
C5: 330 µF/10 V elektrolityczny LOW ESR (PANASONIC_F)
C7: 10 nF ceramiczny X7R (SMD0805)

Półprzewodniki:
U1: TC1047A (SOT-23)
U2: ATtiny25 (SOIC-8)
U3: A8498 (SOIC-8)
D1...D25: dioda LED APA-102 (PLCC-6 5050)
D26: B320A-13-F (SMA)

Pozostałe:
P1: potencjometr montażowy SMD 10 kΩ typu 3305SMD-10K SR PASSIVES (SMD)
L1: dławik mocy SMD 33 µH typu DE1207-33 (DE1207)
L2: dławik SMD 10 µH (SMD0805)
MODE: microswitch TACT SMD (DTSM-3)

Ustawienia Fuse-bitów:
CKSEL3...0: 0010
SUT1...0: 10
CKDIV8: 0
CKOUT: 1
DWEN: 1
EESAVE: 0
RSTDISBL: 1

Rysunek 4. Konstrukcja ramki transmisji danych użytecznych diody APA102

Wiemy już, jak wygląda transmisja pojedynczego bajta, więc pora na pokazanie całej ramki transmisji, której to wygląd pokazano na **rysunku 3**. Jak widać ramka transmisji diody APA102 (a właściwie łańcucha diod tego typu) rozpoczyna się od przesłania słowa o wartości 0x00000000 (4 bajtów o wartości 0x00). Wynika to z faktu, iż dioda APA102 po włączeniu zasilania oczekuje na słowo tego rodzaju by przygotować swój sterownik na odbiór użytecznych danych (przeznaczonych dla siebie) i retransmisję kolejnych sygnałów sterujących na swoje wyjścia (dla kolejnych elementów w łańcuchu). Do tego czasu sterownik każdej z diod tego typu (połączonych w łańcuch) jest niejako przeźroczysty dla nadchodzących danych, co oznacza, że dane (jak i zegar) dostępne są na ten moment dla każdej z diod w łańcuchu. Po odbiorze słowa startowego, de facto synchronizującego całą transmisję, przesyłane są użyteczne dane (4 bajty danych koloru i jasności świecenia) dla kolejnych diod w łańcuchu aż do wystąpienia ramki kończącej transmisję o wartości 0xFFFFFFFF (4 bajtów o wartości 0xFF).

W tym miejscu zapewne zadacie sobie pytanie skąd każda z diod w łańcuchu wie, które z przesyłanych danych użytecznych przeznaczone są właśnie dla niej, a nie dla innej? O tym za chwilę. Popatrzcie w tym momencie na **rysunek 4**, gdzie pokazano wygląd ramki transmisji danych użytecznych diody APA102. Jak widać, pierwszy bajt danych definiuje jasność świecenia każdej diody LED (ostatnich 5-bitów tego bajta, gdyż pierwsze zawsze powinny wynosić 0b111), zaś następujące po nim 3 bajty danych definiują udział poszczególnych składowych koloru diody LED w kolejności B, G, R i rozdzielczości 8-bitów dla każdego z nich. Transmisja, zgodnie z tym co napisano powyżej, kończy się ramką końcową złożoną z 4 bajtów o wartości 0xFF. Przynajmniej w teorii. Tak naprawdę konstrukcja ramki kończącej transmisję powinna być nieco inna, gdyż przesłanie 4 bajtów o wartości 0xFF sprawdzi się wyłącznie wtedy, gdy w łańcuchu mamy nie więcej niż 65 diod tego typu. Producent układu niezbyt dokładnie przyłożył się do wytłumaczenia tej kwestii, ale o tym później. Prawda, że proste?

Niemniej jednak, i co widać na rysunku 3, przyjęty sposób transmisji stanowiący podstawę adresowania poszczególnych diod LED w łańcuchu powoduje, iż nie da się zaadresować (czyli przesłać do niej danych) na przykład diody czwartej w łańcuchu bez przesłania wcześniejszych (i zdawałoby się niepotrzebnych w tym momencie) danych dla diody trzeciej, drugiej i pierwszej. Mimo tego ograniczenia i z uwagi na bardzo dużą, dopuszczalną prędkość zegara transmisji (15 MHz) nawet przy 1024 diodach w łańcuchu osiągniemy niebywałą, jak na liczbę diod, wartość odświeżania (*frame rate*) na poziomie 230 Hz o czym moglibyśmy pomarzyć stosując diody z rodziny WS2811/WS2812.

A teraz odpowiedź na nurtujące, jak przypuszczam, Was pytanie. Skąd każda z diod w łańcuchu wie, które z przesyłanych danych użytecznych przeznaczone są właśnie dla niej, a nie dla innej, skoro w ramkach danych nie jest zaszyty adres diody LED? Zresztą, jak na dobrą sprawę mielibyśmy ten adres ustawić każdej diodzie? Inżynierowie rozwiązali to w sposób bardzo prosty i zarazem skuteczny. Po odbiorze startowej ramki danych (dostępnej dla każdej z diod w łańcuchu) każda z diod LED w łańcuchu oczekuje na 4 bajty użytecznych danych rozpoczynające się od 3 bitów o wartości „1” (informacji o kolorze i jasności). Do tego czasu nie przekazuje ona żadnych danych ze swojego wejścia na wyjście, czyli jest niejako nieprzeźroczysta dla medium transmisyjnego. Żeby być dokładnym, to tak naprawdę retransmituje ona sygnał zegarowy (zresztą zanegowany), zaś sygnał danych pozostaje na poziomie logicznego 0. Skoro sygnał wyjściowy tejże diody pozostaje na poziomie logicznego 0 to kolejna dioda w łańcuchu nie odbiera tych danych (mimo obecności sygnału zegarowego) jako danych użytecznych, bo jak pamiętamy, dane użyteczne zaczynać się muszą sekwencją 0b111.

Po odebraniu wspomnianych 4 bajtów użytecznych danych bieżąca dioda zapamiętuje je, wyświetla i czyni swój interfejs komunikacyjny przeźroczysty dla kolejno nadchodzących danych, czyli przekazuje je z wejścia na wyjście. Biorąc pod uwagę, iż dokładnie tak zachowuje się każda dioda w łańcuchu dość szybko zdamy sobie sprawę, że kolejne dane użyteczne przesyłane przez tak skonstruowany interfejs komunikacyjny trafiają kolejno do następujących po sobie (w sensie elektrycznym) diod w łańcuchu. Skuteczne i zarazem genialne w swojej prostocie, nieprawdaż?

Ale jest pewien haczyk, o którym warto wspomnieć. Inżynierowie projektujący interfejs komunikacyjny diod APA102 musieli zmierzyć się z problemem przesyłania danych do kolejnych diod LED w łańcuchu w zakresie ich odpowiedniej synchronizacji i wykluczenia zjawiska hazardu. Wykonali to w taki sposób, iż dane dla kolejnej diody LED w łańcuchu są opóźnione o połowę czasu trwania cyklu zegara taktującego, co zrealizowano poprzez zanegowanie tegoż zegara taktującego na wyjściu każdej diody LED. Rozwiązanie proste i skuteczne, ale ma pewną wadę. Skoro dane zatrzaskiwane są przez diodę LED na rosnącym zboczu sygnału to znaczy, że kolejna dioda w łańcuchu musi otrzymać jedno zbocze zegara więcej, by zatrzasnąć dane przeznaczone dla niej a wynika to z zanegowania wyjściowego sygnału zegarowego diody poprzedzającej. Bez tego dodatkowego rosnącego zbocza sygnału każda kolejna dioda w łańcuchu nie wyświetli danych przeznaczonych dla niej a co więcej, jeśli wyślemy dane dla mniejszej ilości diod, niż rzeczywiście znajdują się w łańcuchu i zakończymy je standardową ramką 0xFFFFFFFF to któraś z diod w tymże łańcuchu zaświeci się z maksymalną wartością danych użytecznych (czyli wyświetli kolor biały z maksymalną jasnością).

Niestety producent elementów ograniczył się do enigmatycznych 4 bajtów w rodzaju 0xFF na końcu ramki transmisji nie tłumacząc szczegółowo tego zagadnienia. Reasumując, tak naprawdę końcowa ramka transmisji powinna składać się z n-1 dodatkowych, zboczy sygnału zegarowego, gdzie n oznacza liczbę diod w łańcuchu. Znaczy to ni mniej, ni więcej, że dla 2 diod w łańcuchu konieczne jest przesłanie 1 dodatkowego zbocza sygnału, dla 3 diod, 2 zboczy sygnału i tak dalej. Jako, że wspomniane zbocza sygnału przesyłamy partiami (po 8 bitów, czyli 16 zboczy), bo transmitujemy pełne bajty danych w ramach tejże ramki, to rzeczywista liczba transmitowanych bajtów 0xFF wynika z prostej zależności w rodzaju: (n+14)/16 i to jest właśnie właściwa konstrukcja końcowej ramki danych!

Szkoda, że to zagadnienie trzeba było odkrywać samemu zamiast opierać się na dokumentacji producenta. Tyle w kwestii naszych ciekawych elementów LED. Zanim pokażę niezmiernie prosty schemat termometru, kilka niezbędnych szczegółów implementacyjnych związanych z programową realizacją interfejsu tego typu.

Sterowanie diodami APA102

Na początek, na **listingu 1** pokazano niezbędne definicje z pliku nagłówkowego upraszczającego dostęp do portów oraz porządkującego późniejszy kod. Dalej, na **listingu 2** pokazano niezmiernie

Listing 1. Plik nagłówkowy modułu obsługi diod APA102

```
#define APA102_DDR DDRB
#define APA102_PORT PORTB
#define APA102_CLK PB1
#define APA102_DATA PB2
#define APA102_CLK_1 APA102_PORT |= (1<<APA102_CLK)
#define APA102_CLK_0 APA102_PORT &= ~(1<<APA102_CLK)
#define APA102_DATA_1 APA102_PORT |= (1<<APA102_DATA)
#define APA102_DATA_0 APA102_PORT &= ~(1<<APA102_DATA)
#define APA102_AS_OUTPUTS APA102_DDR |= (1<<APA102_CLK)|(1<<APA102_DATA)
```

Listing 2. Kod funkcji inicjalizacyjnej interfejsu komunikacyjnego diod APA102

```
void APA102Init(void){
    //Porty sterujące (CLK, DATA) jako wyjściowe
    //ze stanami spoczynkowymi = 0
    APA102_AS_OUTPUTS;
}
```

Listing 3. Kod funkcji odpowiedzialnej za przesłanie bajta danych przy pomocy interfejsu komunikacyjnego diody APA102

```
void APA102SendByte(uint8_t Byte){
    for(uint8_t i=0; i<8; ++i){
        if(Byte & 0x80) APA102_DATA_1;
        else APA102_DATA_0;
        APA102_CLK_1;
        Byte <<= 1;
        APA102_CLK_0;
    }
}
```

Listing 4. Kod funkcji odpowiedzialnej za przesłanie ramki startowej

```
void APA102StartTrans(void){
    //Start Frame
    APA102SendByte(0x00);
    APA102SendByte(0x00);
    APA102SendByte(0x00);
    APA102SendByte(0x00);
}
```

Listing 5. Kod funkcji odpowiedzialnej za przesłanie ramki danych użytecznych

```
void APA102SendRGBdata(uint8_t Brightness, uint8_t R, uint8_t G, uint8_t B){
    //Brightness
    APA102SendByte(0b11100000|Brightness );
    APA102SendByte(B);
    APA102SendByte(G);
    APA102SendByte(R);
}
```

Listing 6. Kod funkcji odpowiedzialnej za przesłanie ramki kończącej transmisję

```
void APA102StopTrans(uint16_t numberOfLeds){
    //End Frame
    for (uint16_t i = 0; i < (14+numberOfLeds)/16; ++i) APA102SendByte(0xFF);
}
```

prosty kod funkcji inicjalizacyjnej interfejsu komunikacyjnego, którego wyłącznym zadaniem jest ustawienie kierunku portów danych i zegara oraz ich stanów spoczynkowych (0). Na **listingu 3** pokazano z kolei kod funkcji odpowiedzialnej za przesłanie bajta danych przy pomocy interfejsu komunikacyjnego.

Jak widać, w pokazanym kodzie nie są stosowane żadne opóźnienia w rodzaju funkcji *_delay_us*, gdyż wspomniany interfejs komunikacyjny udostępnia wysokie prędkości zegara rzędu 15 MHz, przez co w przypadku naszego mikrokontrolera taktowanego zegarem o częstotliwości 1 MHz nie zachodzi możliwość przetaktowania medium transmisyjnego. Niemniej jednak w przypadku implementacji na szybkich procesorach z rodziny ARM taka ewentualność musi być brana pod uwagę. Inna sprawa, że do implementacji tego rodzaju medium transmisyjnego można użyć sprzętowy sprzęg SPI znajdujący się na pokładzie większości mikrokontrolerów, ale nawet wtedy to ograniczenie jest nadal wiążące.

Dalej, na **listingu 4** pokazano kod funkcji odpowiedzialnej za przesłanie ramki startowej, zaś na **listingu 5** pokazano z kolei kod funkcji odpowiedzialnej za przesłanie ramki danych użytecznych. I na sam koniec, na **listingu 6** pokazano kod funkcji odpowiedzialnej za przesłanie ramki kończącej transmisję, której ciało uwzględnia uwagi opisane wcześniej, więc jest niejako rozwinięciem dokumentacji producenta. Tyle w kwestiach implementacyjnych.

Budowa i działanie

Schemat ideowy naszego urządzenia pokazano na **rysunku 5**. Jak widać, zaprojektowano bardzo prosty, wręcz trywialny, system mikroprocesorowy, którego sercem jest niewielki mikrokontroler firmy Microchip (dawniej Atmel) o oznaczeniu ATtiny25 taktowany wewnętrznym oscylatorem 1 MHz. Jest odpowiedzialny za realizację całej, założonej funkcjonalności.

Jako element pomiarowy (termometr) zastosowano scalony przetwornik temperatura-napięcie pod postacią układu scalonego TC1047A produkcji Microchip cechujący się doskonałą liniowością (z nachyleniem 10 mV/°C) oraz dość dużą dokładnością. Napięcie wyjściowe z termometru scalonego, jak i napięcie z dzielnika napięciowego zbudowanego z elementów P1/R1 mierzone są poprzez wbudowany w strukturę mikrokontrolera 10-bitowy przetwornik ADC i przeliczane na odpowiednie wartości wykorzystywane w programie obsługi aplikacji.

Wspomniany dzielnik napięcia zapewnia sprzętowo-programową realizację regulacji intensywności świecenia diod LED RGB, których pracą steruje nasz mikrokontroler przy udziale, pokazanej wcześniej, programowej realizacji interfejsu SPI. Ostatnim elementem obsługiwanym przez nasz mikrokontroler jest mikroprzełącznik MODE pozwalający, jak łatwo się domyślić, na zmianę sposobu prezentacji temperatury pomiędzy trybami RGB a COLOR, o czym wspomniano już wcześniej. Stosowne ustawienie zapisywane jest w nieulotnej pamięci EEPROM mikrokontrolera przez co pamiętane jest nawet po wyłączeniu zasilania.

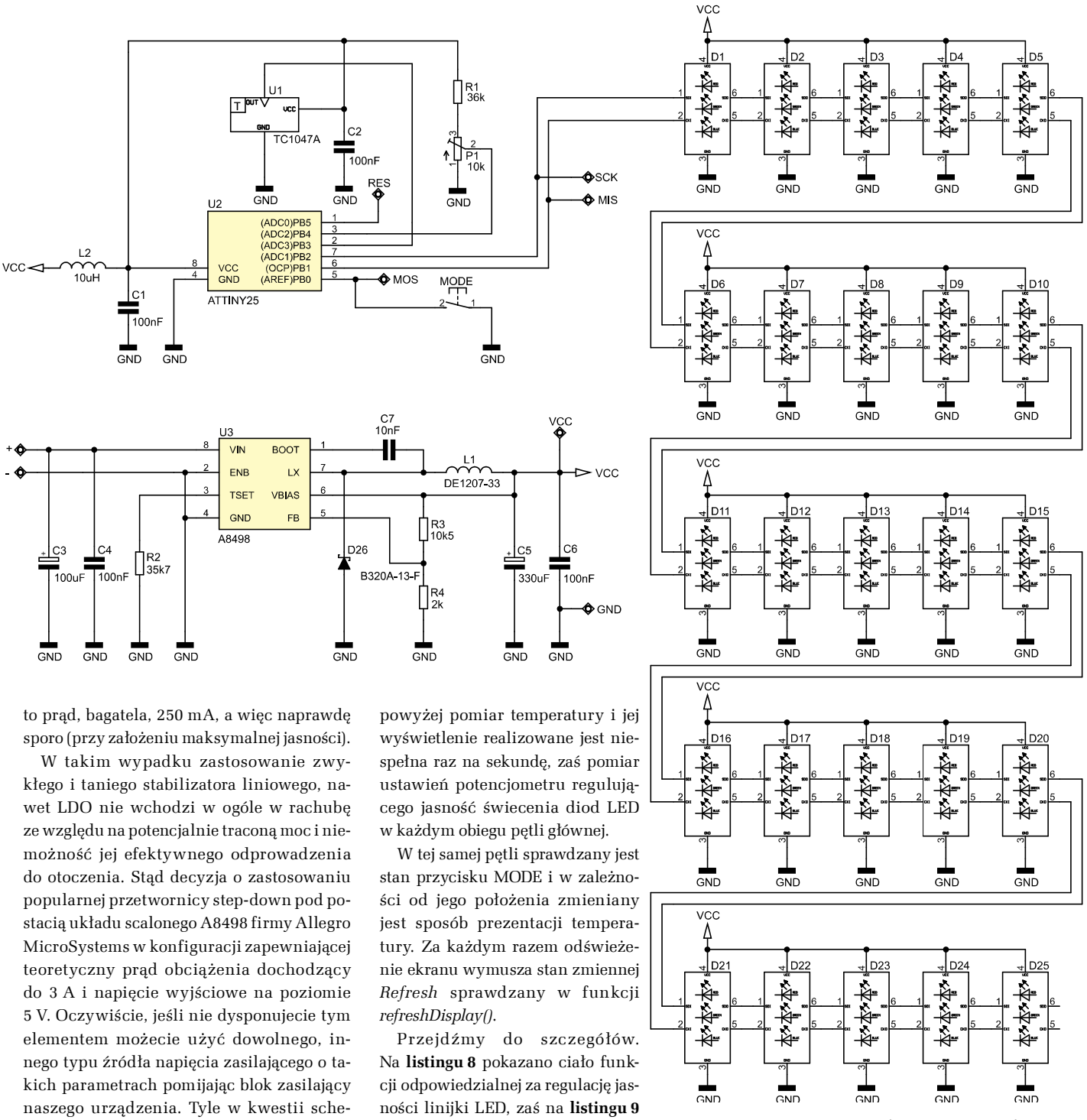
Uważny Czytelnik dostrzeże dość rozbudowany blok zasilający. Nie bez przyczyny zdecydowałem się na wykorzystanie przetwornicy typu step-down o dużej (86%) sprawności. Otóż każda z diod LED w przypadku skonfigurowania ich do pracy z maksymalną jasnością i maksymalnym udziałem każdej ze składowych RGB (świecąca wtedy de facto na biało) pobierać będzie ze źródła zasilania prąd rzędu 60 mA. Co prawda w naszym urządzeniu żadna z diod LED nie pracuje z nastawami 100% dla każdej ze składowych koloru, lecz i tak zgrubnie można przyjąć, iż każda z nich pobierze ze źródła zasilania prąd na poziomie 10 mA. Przy 25 świecących w tym samym czasie diodach LED daje

Listing 7. Kod funkcji *main* aplikacji termometruRGB

```
int main(void){
    uint8_t Timer = 0;

    //Podciągnięcie przycisku MODE pod VCC
    MODE_PULLED_UP;
    //Inicjalizacja portów sterujących diodami LED
    APA102Init();
    //Proste demo na wstępie
    Demo();

    while(1){
        //Co około sekundę mierzymy
        //i wyświetlamy temperaturę
        if((++Timer%128) == 0) Refresh = 1;
        //Obsługa klawiatury zmieniającej sposób
        //prezentacji temperatury
        handleKeyboard();
        //Obsługa potencjometru odpowiedzialnego
        //za regulację jasności linijki LED
        readBrightness();  //Czas wykonania ok. 7 ms
        //Odświeżenie ekranu, czyli wyświetlenie
        //bieżącej temperatury według
        //bieżących kryteriów wyświetlania
        refreshDisplay();
    }
}
```



Rysunek 5. Schemat ideowy urządzenia

to prąd, bagatela, 250 mA, a więc naprawdę sporo (przy założeniu maksymalnej jasności).

W takim wypadku zastosowanie zwykłego i taniego stabilizatora liniowego, nawet LDO nie wchodzi w ogóle w rachubę ze względu na potencjalnie traconą moc i niemożność jej efektywnego odprowadzenia do otoczenia. Stąd decyzja o zastosowaniu popularnej przetwornicy step-down pod postacią układu scalonego A8498 firmy Allegro MicroSystems w konfiguracji zapewniającej teoretyczny prąd obciążenia dochodzący do 3 A i napięcie wyjściowe na poziomie 5 V. Oczywiście, jeśli nie dysponujecie tym elementem możecie użyć dowolnego, innego typu źródła napięcia zasilającego o takich parametrach pomijając blok zasilający naszego urządzenia. Tyle w kwestii schematu ideowego urządzenia.

Program sterujący

Zanim przejdę do szczegółów konstrukcyjnych kilka niezbędnych słów na temat programu głównego aplikacji, którego ciało (wraz z deklaracją niezbędnych zmiennych globalnych) pokazano na **listingu 7**. Jak widać jest on niezmiernie prosty i klarowny. Z uwagi na prostotę aplikacji nie korzystałem z timerów sprzętowych, tylko zaprzągłem zwykłą zmienną *uint8_t Timer* do generowania stosownych opóźnień programowych korzystając z faktu, że wywołanie funkcji obsługi potencjometru odpowiedzialnego za regulację jasności linijki LED zajmuje ok. 7 ms czasu mikrokontrolera. Zgodnie z kodem pokazanym powyżej pomiar temperatury i jej wyświetlenie realizowane jest niespełna raz na sekundę, zaś pomiar ustawień potencjometru regulującego jasność świecenia diod LED w każdym obiegu pętli głównej.

W tej samej pętli sprawdzany jest stan przycisku MODE i w zależności od jego położenia zmieniany jest sposób prezentacji temperatury. Za każdym razem odświeżenie ekranu wymusza stan zmiennej *Refresh* sprawdzany w funkcji *refreshDisplay()*.

Przejdźmy do szczegółów. Na **listingu 8** pokazano ciało funkcji odpowiedzialnej za regulację jasności linijki LED, zaś na **listingu 9** pokazano z kolei ciało funkcji odpowiedzialnej za odświeżanie ekranu. Jak widać, i o czym wspomniano wcześniej, wprowadzono 2 tryby wyświetlania informacji o temperaturze: RGB i COLOR. Kolor każdej z diod LED definiowany jest poprzez wartości tablic reprezentujących poszczególne składowe koloru (R, G i B) i wynika z bieżącej temperatury otoczenia. Definicje tablic kolorów pokazano na **listingu 10**, zaś na **rysunku 6** pokazano udział poszczególnych składowych koloru (R, G, B) w wynikowym kolorze diody LED w zależności od mierzonej temperatury otoczenia.

Na koniec, na **listingu 11** pokazano ciało funkcji (wraz z odpowiednimi definicjami) odpowiedzialnej za pomiary przy użyciu wbudowanego przetwornika ADC. Tyle w kwestiach implementacyjnych. Przejdźmy do zagadnień montażowych.

Montaż i uruchomienie

Schemat montażowy naszego urządzenia pokazano na **rysunku 7**. Jak widać, zaprojektowano bardzo zwarty, dwustronny obwód drukowany przeznaczony do montażu powierzchniowego (po obu stronach laminatu) wyglądem przypominający analogowy termometr pokojowy.

Muszę przyznać, iż mimo prostoty schematu ideowego, projekt tej płytki był dla mnie niezłym wyzwaniem. Z uwagi na dość duży prąd szczytowy przetwornicy zasilającej

Listing 8. Kod funkcji odpowiedzialnej za regulację jasności linijki LED

```
void readBrightness(void){
    static uint8_t lastBrightness;

    Brightness = readADC(BRIGHTNESS);
    if(lastBrightness != Brightness){
        lastBrightness = Brightness;
        Refresh = 1;
    }
}
```

Listing 9. Kod funkcji odpowiedzialnej za odświeżanie ekranu

```
void refreshDisplay(void){
    uint8_t Temperature;  //0...24

    if(Refresh){
        Refresh = 0;

        Temperature = readADC(TEMPERATURE);  //0...24

        //W zależności od sposobu prezentacji temperatury
        //wysyłamy odpowiednie wartości do diod APA102
        APA102StartTrans();
        if(eeprom_read_byte(&Mode) != MODE_COLOR){  //RGB
            for(uint8_t i=0; i<25; ++i)
                if(i <= Temperature) APA102SendRGBdata(Brightness, R[i], G[i], B[i]);
                else APA102SendRGBdata(0, 0, 0, 0);
        } else {  //COLOR
            for(uint8_t i=0; i<25; ++i)
                if(i <= Temperature) APA102SendRGBdata(Brightness, R[Temperature], G[Temperature], B[Temperature]);
                else APA102SendRGBdata(0, 0, 0, 0);
        }
        APA102StopTrans(25);

        //Dodane wyłącznie z uwagi na prosty sposób
        //eliminacji drgania styków w funkcji handleKeyboard()
        _delay_ms(35);
    }
}
```

Listing 10. Definicje tablic kolorów diod LED

```
//Stałe składowych RGB dla kolejnych wartości mierzonej temperatury (10...34 'C)
uint8_t B[25] = {255, 234, 213, 192, 171, 150, 129, 108, 87, 66, 45, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8_t G[25] = {0, 21, 42, 63, 84, 105, 126, 147, 168, 189, 210, 231, 255, 231, 210, 189, 168, 147, 126, 105, 84, 63, 42, 21, 0};
uint8_t R[25] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 45, 66, 87, 108, 129, 150, 171, 192, 213, 234, 255};
```

Listing 11. Ciało funkcji (wraz z odpowiednimi definicjami) odpowiedzialnej za pomiary przy użyciu wbudowanego przetwornika ADC

```
//Definicja kanałów ADC
#define TEMPERATURE 0x03
#define BRIGHTNESS 0x02

//Przeliczone wartości ADC dla temperatur od 10'C...34'C (ostatnia wartość dodana z uwagi na mechanizm wyszukiwania indexu w funkcji ADC)
const uint16_t tc1047TransferFunction[26] PROGMEM = {563, 572, 581, 591, 600, 609, 619, 628, 637, 646, 656, 665, 674, 684, 693, 702, 712,
721, 730, 740, 749, 758, 768, 777, 786, 1023};

uint8_t readADC(uint8_t Channel){  //6.8 ms
    uint8_t Index;
    uint16_t adcVal = 0, currVal, nextVal;


    //Konfiguracja przetwornika ADC: Referencja = 1.1V,
    //wejście specyfikowane wartością zmiennej Channel
    ADMUX = (1<<REFS1)|Channel;

    //Wykonanie 32 pomiarów ADC
    //w celu późniejszego uśrednienia wartości wynikowej
    for(uint8_t i=0; i<32; ++i){
        //Start konwersji, Prescaler= 16 (62.5 kHz)
        ADCSRA = (1<<ADEN)|(1<<ADSC)|(1<<ADPS2);
        //Czekamy na jej zakończenie – 210us
        while(ADCSRA & (1<<ADSC));
        adcVal += ADC;
    }

    //Uśrednienie wartości wynikowej
    adcVal /= 32;

    //W zależności od mierzonego kanału wartość funkcji
    //liczona jest według innego algorytmu
    if(Channel == BRIGHTNESS){
        return adcVal>>5;  //Zakres 0...31
    } else { //TEMPERATURE
        //Ograniczamy wartość od dołu
        if(adcVal < 558) adcVal = 558;

        //Wyszukujemy wartość napięcia (jego index) w tablicy TC1047
        for(Index = 0; Index < 25; ++Index){
            currVal = pgm_read_word(&tc1047TransferFunction[Index]);
            nextVal = pgm_read_word(&tc1047TransferFunction[Index+1]);
            if(adcVal >= currVal && adcVal < nextVal) break;
        }
        return Index;
    }
}
```

diody LED musiałem zadbać o odpowiednie prowadzenie sygnałów krytycznych (zwłaszcza w bloku przetwornicy step-down) oraz masy zasilania. Warto podkreślić, iż nieodpowiednie prowadzenie ścieżek zasilających diody LED mogłoby doprowadzić do powstania dość dużych indukcyjności pasożytniczych co przy sterowaniu PWM w konsekwencji prowadziłoby do powstawania przepięć mogących uszkodzić elementy półprzewodnikowe.

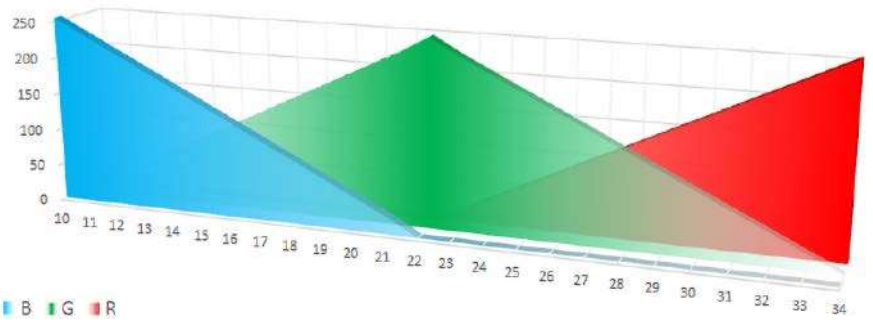
Montaż urządzenia rozpoczynamy od warstwy TOP, gdzie w pierwszej kolejności przylutowujemy diody RGB. Aby ułatwić sobie nieco to zadanie wspomniane elementy możemy wcześniej przykleić do obwodu drukowanego stosując kropelkę dowolnego kleju nieprzewodzącego prąd. Warto zwrócić uwagę, iż każda kolejna dioda obrócona jest w stosunku do diody ją poprzedzającej o kąt 180°.

W następnej kolejności lutujemy mikrokontroler a później elementy pasywne i przycisk MODE. W tym momencie przechodzimy na warstwę BOTTOM, gdzie w pierwszej kolejności lutujemy elementy półprzewodnikowe, zaś na samym końcu elementy pasywne.

Poprawnie zmontowany układ nie wymaga jakichkolwiek regulacji i powinien działać tuż po włączeniu zasilania. Na **fotografii 1** pokazano wygląd zmontowanej płytki drukowanej urządzenia widzianej od strony TOP (w wersji prototypowej), zaś na **fotografii 2** odpowiedni

widok od strony BOTTOM (także w wersji prototypowej).

Na **rysunku 8** pokazano z kolei szablon (w skali 1:1) płyty czołowej termometru, który możemy wydrukować na papierze technicznym (o większej gramaturze) i zamocować ponad obwodem drukowanym urządzenia. Z uwagi na fakt, iż soczewki diod LED są w pełni przezroczyste, dla poprawy efektu wizualnego i w celu rozmycia składowych barw poszczególnych diod LED pod wspomnianym szablonem podkleić możemy kawałek mlecznej folii rozpraszającej światło.

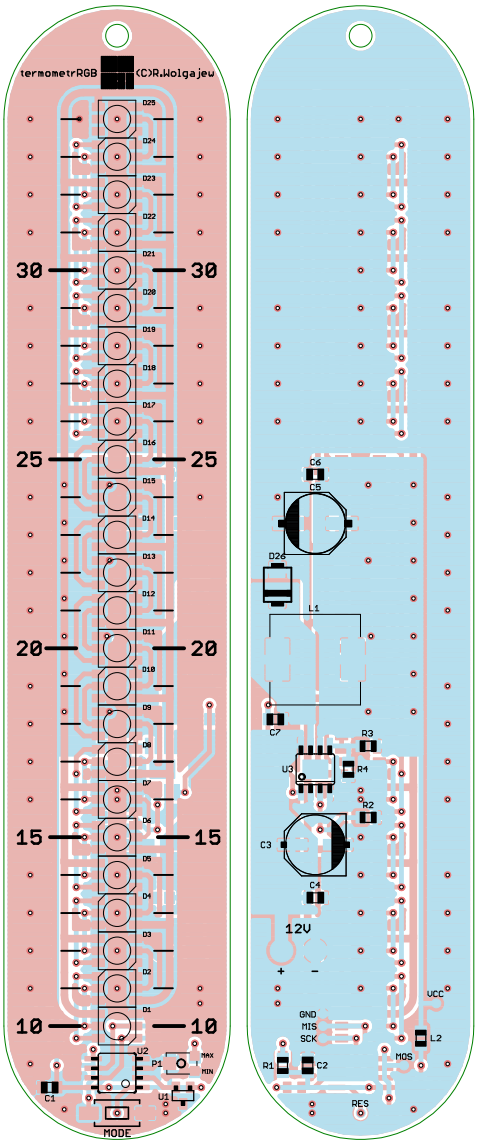


Rysunek 6. Udział poszczególnych składowych koloru (R, G, B) w wynikowym kolorze diody LED w zależności od mierzonej temperatury otoczenia

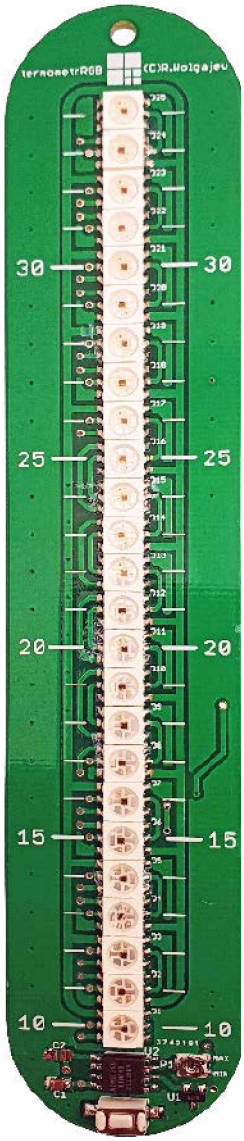
Podsumowanie

Na koniec dodam, iż po włączeniu urządzenia prezentowana jest prosta, acz efektowna animacja w postaci linijki w kolorach widma światła, która jednocześnie pełni funkcję testu diod LED. Dla osób, które we własnym zakresie będą chciały poeksperymentować z ustawieniami urządzenia czy też dopasować je do własnych wymagań, zamieszczam pełny kod aplikacji w postaci projektu (spakowanego) środowiska Eclipse. Z uwag praktycznych, w dobie problemów z dostępnością elementów półprzewodnikowych, podpowiem, iż w przypadku braku w hurtowniach podzespołów zastosowanych w niniejszym urządzeniu warto zajrzeć na strony producentów, gdzie niejednokrotnie zamówić możemy darmowe próbki elementów, z czego i ja korzystałem podczas implementacji naszego urządzenia.

Robert Wołgajew, EP

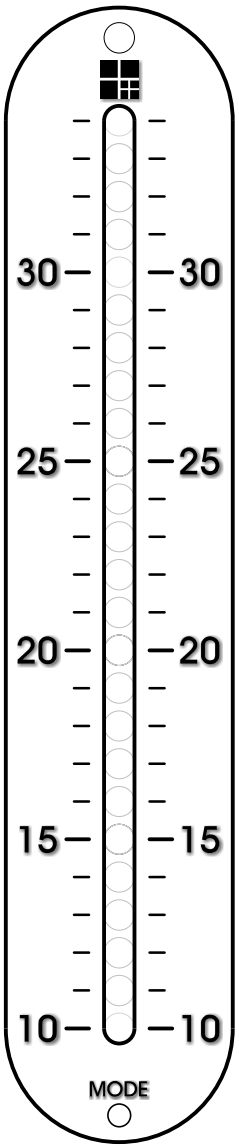


Rysunek 7. Schemat montażowy płytki PCB



Fotografia 1. Wygląd zmontowanej płytki drukowanej urządzenia widzianej od strony TOP (w wersji prototypowej)

Fotografia 2. Wygląd zmontowanej płytki drukowanej urządzenia widzianej od strony BOTTOM (w wersji prototypowej)



Rysunek 8. Szablon (w skali 1:1) płyty czołowej termometru