W ofercie AVT*
AVT5982

Podstawowe parametry:
- obrazuje poziom sygnału na dwóch linijkach, z których każda składa się z 20 diod LED,
- 3 tryby wyświetlania: wskaźnik punktowy, wskaźnik liniowy (bargraph), wskaźnik liniowy (bargraph) z pamięcią wartości szczytowych,
- maksymalne napięcie wejściowe (RMS): 1,4 V,
- poziomy odniesienia dla poszczególnych diod LED w przypadku skali domyślnej (dBu): –32, –30, –28, –26, –24, –22, –20, –18, –16, –14, –12, –10, –8, –6, –4, –2, 0, 1, 3, 5,
- zasilanie: 8...10 V, 170 mA.

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

AVT5866	Spectra – analizator widma sygnału audio (EP 6/2021)
AVT5767	Stereofoniczny wskaźnik poziomu wysterowania z funkcją Peak-Hold (EP 5/2020)
AVT5748	SpectrumDFT – analizator widma sygnału akustycznego (EP 3/2020)
AVT5712	Spectrum – prosty analizator widma sygnału akustycznego (EP 9/2019)
AVT5678	Stereofoniczny wskaźnik wysterowania (EP 6/2019)
AVT5585	Sterownik wskaźnika wychyłowego do wzmacniacza (EP 1/2018)
AVT1716	Wskaźnik wysterowania z pamięcią (EP 12/2012)
AVT1517	Wskaźnik nie tylko wysterowania (EP 9/2012)
AVT5219	Wizualizator do Winampa na USB (EP 1/2010)
AVT5210	Analizator widma sygnału audio (EP 11/2009)

* **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.

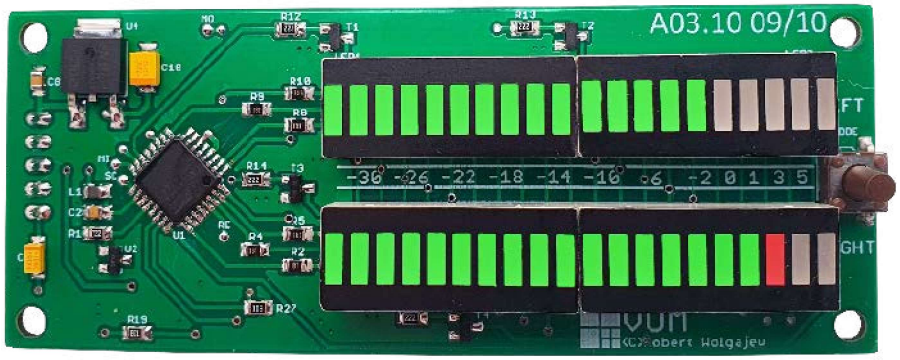
Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

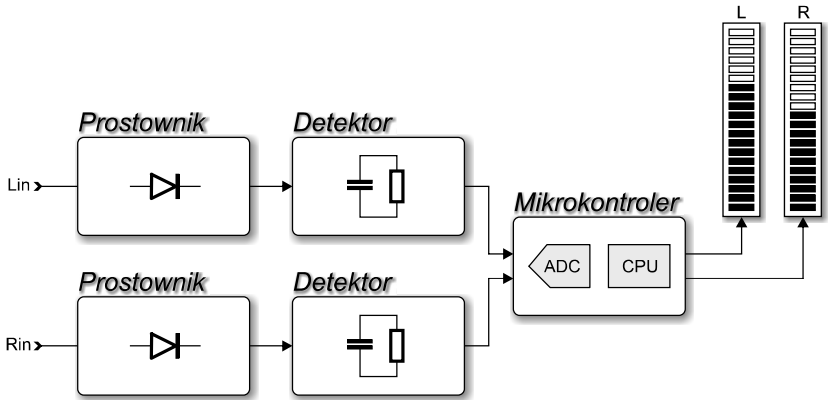
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

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

VUM – mikroprocesorowy wskaźnik wysterowania sygnału audio

Projekt mikroprocesorowego wskaźnika wysterowania sygnału audio chodził za mną od dłuższego czasu, ponieważ z rozrzewnieniem wspominam moje pierwsze przygody z elektroniką w postaci kultowego (i trudnego do zdobycia) na owe czasy układu typu UL1970/UL1980 produkcji CEMI (będącego kopią rozwiązań zagranicznych). Pozwalając sobie na odrobinę prywaty w stylu off-topic, muszę przyznać, iż były to piękne czasy, gdzie mimo ogólnych problemów ze zdobyciem czegokolwiek (nie tylko elementów elektronicznych) każde skonstruowane urządzenie cieszyło po tysiąckroć. Teraz mamy dużo większe możliwości i urządzenia tego typu są ograniczone tylko pomysłowością konstruktora.





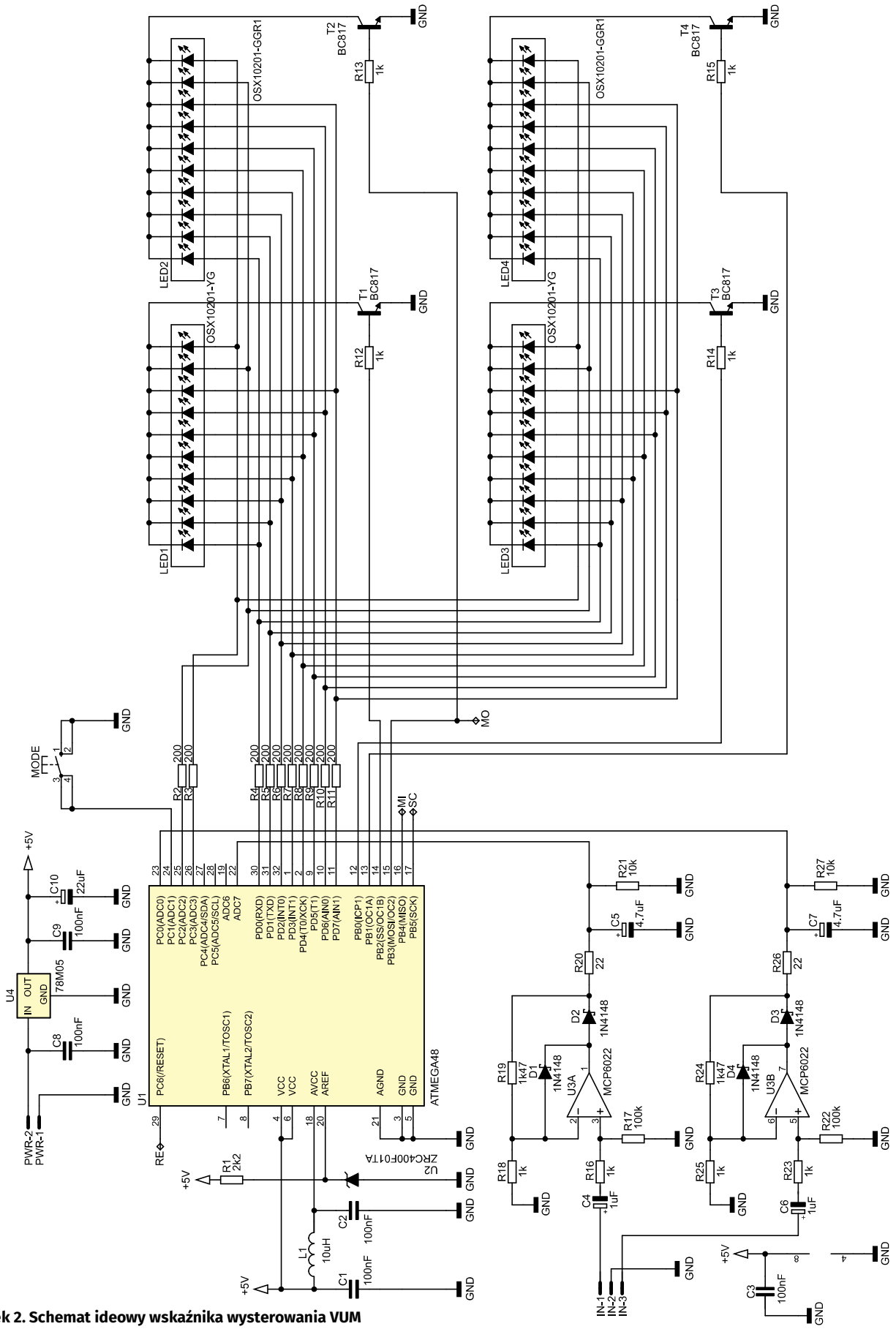
Rysunek 1. Schemat blokowy wskaźnika wysterowania VUM

Dzisiaj, gdy wszystko jest na wyciągnięcie ręki, możemy przebierać w dostępnych rozwiązaniach układowych i mimo, że mamy czasy wszechobecnej techniki cyfrowej, to nadal można zakupić wiele analogowych układów wskaźników wysterowania różnej maści (w tym dość egzotycznych układów z Azji). Niemniej jednak implementacja analogowego wskaźnika tego typu nie jest żadnym wyzwaniem, gdyż aplikacje takich układów są niezmiernie proste i nie pozostawiają płaszczyzny do własnej inwencji a na dodatek mają z góry ustaloną funkcjonalność. Właśnie stąd wziął się pomysł na skonstruowanie własnej wersji tego typu urządzenia o rozszerzonej funkcjonalności i możliwości dostosowania do własnych potrzeb.

Zastanówmy się na początek, jak zbudowane powinno być takie urządzenie. Na pewno niezbędny jest jakiś obwód wejściowy przygotowujący sygnał audio do digitalizacji, następnie przetwornik ADC a na końcu mikrokontroler i wyświetlacz. Idea wydaje się prosta. Schemat blokowy urządzenia VUM pokazano na **rysunku 1**.

Poziomy sygnałów audio i stosowne skale

Zanim przejdę do szczegółowych rozwiązań układowych, kilka niezbędnych słów na temat poziomów sygnałów audio i stosownych skal. Niestety nie mam dobrych wieści. Stosowane są różne skale, dla których przewidziano różne sygnały odniesienia. I tak, mamy poziomy wyrażone w dB, w dBu, w dBV, dBFS (domena urządzeń cyfrowych) i wreszcie VU (skala analogowych mierników wysterowania nazywanych



Rysunek 2. Schemat ideowy wskaźnika wysterowania VUM

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

Rezystory: (SMD0805):
R1: 2.2 kΩ
R2...R11: 200 Ω
R12...R15, R16, R23: 1 kΩ
R18, R25: 1 kΩ 1%
R21, R27: 10 kΩ
R17, R22: 100 kΩ
R19, R24: 1,47 kΩ 1%
R20, R26: 22 Ω

Kondensatory:
C1, C2, C3, C8, C9: 100 nF ceramiczny X7R (SMD0805)
C4, C6: 1 µF/10 V tantalowy (A/3216-18R)
C5, C7: 4,7 µF/10 V tantalowy (A/3216-18R)
C10: 22 µF/10 V tantalowy (B/3528-21W)

Półprzewodniki:
LED1, LED3: linijka diodowa zielona typu OSX10201-YG
LED2, LED4: linijka diodowa zielono-czerwona typu OSX10201-GGR1
D1...D4: 1N4148 (miniMELF)
T1...T4: BC817 (SOT23)
U1: ATmega48 (TQFP32)
U2: ZRC400F01TA (SOT23)
U3: MCP6022 (SO08)
U4: 78M05 (DPAK)

Pozostałe:
L1: dławik 10 µH (SMD0805)
IN: gniazdo męskie proste 3-pin (NSL25-3W)
PWR: gniazdo męskie proste 2-pin (NSL25-2W)
MODE: microswitch SMD typu DTSM-65N-V-B lub podobny

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

Tabela 1. Wartości poszczególnych poziomów sygnału audio i odpowiadające im wartości napięć oraz wartości na wyjściu ADC

dBu [dB]	VRMS [V]	VP-P[V]	VP [V]	Zmierzone [V]	ADC
5	1,377	3,896	1,948	4,084	1021
3	1,094	3,095	1,547	3,997	999
1	0,869	2,458	1,229	3,733	933
0	0,775	2,191	1,095	3,373	843
–2	0,615	1,740	0,870	2,554	638
–4	0,489	1,382	0,691	1,819	455
–6	0,388	1,098	0,549	1,283	321
–8	0,308	0,872	0,436	0,979	245
–10	0,245	0,693	0,346	0,754	188
–12	0,195	0,550	0,275	0,601	150
–14	0,155	0,437	0,219	0,477	119
–16	0,123	0,347	0,174	0,386	97
–18	0,098	0,276	0,138	0,304	76
–20	0,077	0,219	0,110	0,240	60
–22	0,062	0,174	0,087	0,195	49
–24	0,049	0,138	0,069	0,150	38
–26	0,039	0,110	0,055	0,122	30
–28	0,031	0,087	0,044	0,096	24
–30	0,024	0,069	0,035	0,077	19
–32	0,019	0,055	0,028	0,061	15

wolumetrami, które po raz pierwszy skonstruowano w USA w 1940 roku).

O ile pierwsze 4 są ściśle zdefiniowane i można je obliczyć (w wartościach skutecznych napięcia), o tyle skala VU (Volume Unit) ma różne poziomy odniesienia w zależności od standardu, rozgłośni czy kraju. Jedyne, co można o niej powiedzieć z całą pewnością, to to, że rozciąga się od –20 do +3, podczas gdy wartości –20 odpowiada 0%, zaś 0 odpowiada 100% (gdyż wartości procentowe także znajdują się na tej skali). Zwykle wartości 0 odpowiada sygnał o amplitudzie +4 dBu, lecz nie należy zbytnio przywiązywać się do tej wielkości, gdyż jak wspomniałem, występują znaczące różnice w interpretacji tych wartości. Już teraz powiem, że z tego właśnie powodu w naszym urządzeniu zdecydowałem się na skalę dBu (od –32 dB do +5 dB), jako powszechnie używaną w sprzęcie profesjonalnym (przynajmniej starej daty). Dla tej skali poziom 0 dB odpowiada napięciu skutecznemu o wartości 0,775 V, zaś same obliczenia wykonujemy według poniższego wzoru:

$$dBu = 20 \cdot \log \frac{U}{0{,}775}$$

Budowa i działanie

Tyle tytułem wstępu. Przejdźmy zatem do schematu naszego urządzenia, który pokazano na **rysunku 2**. Jak widać, zaprojektowano bardzo prosty system mikroprocesorowy, którego sercem jest niewielki mikrokontroler firmy Microchip (dawniej Atmel) pod postacią układu ATmega48 taktowany wewnętrznym sygnałem zegarowym o częstotliwości 1 MHz realizujący całą, założoną funkcjonalność urządzenia.

Wybór tego konkretnego układu z szerokiej palety układów firmy Microchip wynikał wyłącznie z potrzeby zastosowania elementu o dużej liczbie wyprowadzeń wyposażonego w przetwornik ADC. Ilość pamięci Flash czy RAM nie miała większego znaczenia, gdyż program obsługi aplikacji naszego urządzenia jest niezmiernie prosty i wykorzystuje ułamek dostępnych możliwości. Mikrokontroler steruje akwizycją danych wbudowanego 10-bitowego przetwornika ADC (tylko 10 razy na sekundę), przetwarza zmierzone wartości i na końcu wyświetla je na dwóch linijkach diodowych (oddzielnie dla lewego i prawego kanału), z których każda składa się z 20 diod LED (zbudowanych z dwóch elementów elektronicznych).

Ponieważ mamy do wysterowania aż 40 diod LED, zastosowano dobrze znane rozwiązanie multipleksowania wyświetlanej informacji, gdzie każda z 4 scalonych linijek LED (złożona z 10 diod) sterowana jest 60 razy na sekundę, niwelując zjawisko migotania. W takim rozwiązaniu redukujemy liczbę niezbędnych wyprowadzeń mikrokontrolera z 40 do zaledwie 14 (10 wspólnych anod i 4 wspólnych katod) a samo sterowanie odbywa się w sposób automatyczny (w tle) dzięki wykorzystaniu procedury obsługi przerwania Timera0. Jest

Listing 1. Kod pliku nagłówkowego mechanizmu multipleksowania

```
#define COM_ANODE_LOWER_PORT PORTD
#define COM_ANODE_LOWER_DDR DDRD
#define COM_ANODE_UPPER_PORT PORTC
#define COM_ANODE_UPPER_DDR DDRC
#define COM_ANODE_UPPER_LOW PC2
#define COM_ANODE_UPPER_HIGH PC3

#define COM_CAT_PORT PORTB
#define COM_CAT_DDR DDRB
#define COM_CAT_LEFT_LOWER PB2
#define COM_CAT_LEFT_UPPER PB3
#define COM_CAT_RIGHT_LOWER PB0
#define COM_CAT_RIGHT_UPPER PB1
#define COM_CAT_BLANK COM_CAT_PORT &=
    ~((1<<COM_CAT_LEFT_LOWER)
    |(1<<COM_CAT_LEFT_UPPER)
    |(1<<COM_CAT_RIGHT_LOWER)
    |(1<<COM_CAT_RIGHT_UPPER))

#define LEFT_LED 0x00
#define RIGHT_LED 0x01

//Zmienna przechowująca wartość
//wyświetlaną na obu linijkach LED
extern volatile uint32_t Led[2];
//Zezwolenie na aktualizację zmiennych Led[]
extern volatile uint8_t readyForUpdate;
```

to standardowe i bardzo wygodne rozwiązanie tego rodzaju zagadnień.

Ostatnim elementem obsługiwanym przez nasz mikrokontroler jest przycisk MODE, za pomocą którego wybieramy tryb wyświetlania informacji o poziomie sygnału spośród następujących:

- wskaźnik punktowy,
- wskaźnik liniowy (bargraph),
- wskaźnik liniowy (bargraph) z pamięcią wartości szczytowych.

Wybrane ustawienie zapamiętywane jest w nieulotnej pamięci EEPROM mikrokontrolera i staje się aktywne po włączeniu zasilania.

Warto również zaznaczyć, że przetwornik ADC wbudowany w strukturę mikrokontrolera korzysta z zewnętrznego, scalonego źródła napięcia odniesienia (4,096 V) pod postacią układu ZRC400F01TA, dzięki czemu osiągnięto szeroki zakres przetwarzanych napięć sygnału audio i dużą dokładność.

Skoro mowa o sygnale audio, to warto w skrócie opisać wejściowy obwód kondycjonujący (identycznego dla każdego z kanałów audio), który jest niezbędny z uwagi na charakter szybkozmiennych sygnałów audio. Jak widać, wejściowy sygnał audio trafia na wejście dość nietypowego prostownika jednopołówkowego zbudowanego z wykorzystaniem wzmacniacza operacyjnego (koniecznie typu rail-to-rail input/output) pracującego w konfiguracji nieodwracającej (ze wzmocnieniem równym około 2,47) i zasilanego napięciem niesymetrycznym. Dlaczego napisałem, iż aplikacja wspomnianego prostownika jest dość nietypowa? Wyłącznie z uwagi na fakt, że książkowe układy prostowników operacyjnych korzystają praktycznie zawsze z konfiguracji odwracającej i zasilania symetrycznego, którego chciałem w naszej implementacji uniknąć.

Abstrahując nawet od tego, uważny Czytelnik zauważy z pewnością, iż nawet bez obecności diody na wyjściu wzmacniacza operacyjnego (D3 dla prawego kanału) układ taki i tak pełniłby funkcję prostownika, gdyż wzmacniacz operacyjny zasilany wyłącznie napięciem dodatnim nie przepuści sygnałów o amplitudach ujemnych. Tak, to prawda, ale dioda D3 (jak i D4) ogrywa w tej implementacji inną rolę. Zauważmy, iż na wyjściu prostownika (katoda diody D3) znajduje się prosty układ RC pełniący zarówno funkcję układu całkującego, jak i detektora wartości szczytowej, który dostarcza na wejście przetwornika ADC wbudowanego w strukturę mikrokontrolera w zasadzie napięcie stałe, którego amplituda jest proporcjonalna do chwilowego napięcia szczytowego przebiegu audio. Stałe czasowe tego układu dobrano w taki sposób, by ładowanie kondensatora C7 sygnałem z prostownika przebiegało niemalże natychmiastowo (przez rezystor R26), zaś rozładowanie (przy

Listing 2. Kod funkcji inicjującej mechanizm multipleksowania

```
void initMultiplex(void){
    //Porty wspólnych anod, jako porty wyjściowe ze stanem nieaktywnym “0”
    COM_ANODE_LOWER_DDR = 0xFF;
    COM_ANODE_UPPER_DDR |= (1<<COM_ANODE_UPPER_LOW)
        |(1<<COM_ANODE_UPPER_HIGH);
    //Port wspólnych katod, jako port wyjściowy ze stanem nieaktywnym “0”
    COM_CAT_DDR |= (1<<COM_CAT_LEFT_LOWER)
        |(1<<COM_CAT_LEFT_UPPER)
        |(1<<COM_CAT_RIGHT_LOWER)
        |(1<<COM_CAT_RIGHT_UPPER);

    //Konfiguracja licznika Timer0 w celu generowania przerwania
    //do obsługi multipleksowania linijek LED (240 Hz)
    TCCR0A = (1<<WGM01);    //Tryb CTC
    TCCR0B = (1<<CS02);     //Preskaler = 256 @ 1MHz
    //240 Hz (przerwanie 240 razy na sekundę,
    //60 razy na sekundę dla każdej linijki LED)
    OCR0A = 15;
    //Uruchomienie przerwania Output Compare Match A (od porównania)
    TIMSK0 = (1<<OCIE0A);
}
```

Listing 3. Kod funkcji obsługi przerwania od Timer0, który jest odpowiedzialny za mechanizm multipleksowania

```
ISR(TIMER0_COMPA_vect){
    static uint8_t Nr;  //Numer kolejnej linijki do wyświetlenia
    uint16_t Value = 0; //Optymalizacja dostępu volatile

    readyForUpdate = 0;

    //Wybranie odpowiedniej wartości do pokazania
    switch(Nr){
        case 0: Value = Led[LEFT_LED]; break;
        case 1: Value = Led[LEFT_LED] >> 10; break;
        case 2: Value = Led[RIGHT_LED]; break;
        case 3: Value = Led[RIGHT_LED] >> 10; break;
    }
    //Wygaszenie wspólnych katod
    COM_CAT_BLANK;

    //Wystawienie właściwych stanów na portach wspólnych anod
    COM_ANODE_LOWER_PORT = Value & 0xFF;
    if(Value & 0b100000000)
        COM_ANODE_UPPER_PORT |= (1<<COM_ANODE_UPPER_LOW);
    else COM_ANODE_UPPER_PORT &= ~(1<<COM_ANODE_UPPER_LOW);
    if(Value & 0b1000000000)
        COM_ANODE_UPPER_PORT |= (1<<COM_ANODE_UPPER_HIGH);
    else COM_ANODE_UPPER_PORT &= ~(1<<COM_ANODE_UPPER_HIGH);

    //Załączenie odpowiedniej wspólnej katody
    COM_CAT_PORT |= pgm_read_byte(&comCat[Nr]);

    //Wybranie kolejnej linijki LED
    Nr = (Nr+1) & 0x03;
    //Zezwolenie na aktualizację zmiennych Led[] w pętli głównej
    if(Nr == 0) readyForUpdate = 1;
}
```

każdym spadku sygnału wyjściowego prostownika) następowało o kilka rzędów wielkości wolniej (przez rezystor R27 i R24). Dioda D3 w tym układzie zapobiega rozładowaniu kondensatora wygładzającego C7 przez rezystancję wyjściową wzmacniacza operacyjnego podczas ujemnego półokresu sygnału wejściowego, zaś D4 zapewnia dodatkową ścieżkę rozładowania dla tegoż kondensatora. Jako że wspomniane stałe czasowe są znacząco różne, osiągnięto całkiem rzeczywisty efekt odpowiedzi układu na zmianę napięcia wejściowego sygnału audio symulującą ruchy wskazówki analogowego miernika

Listing 4. Kod funkcji odpowiedzialnej za inicjalizację wbudowanego przetwornika ADC

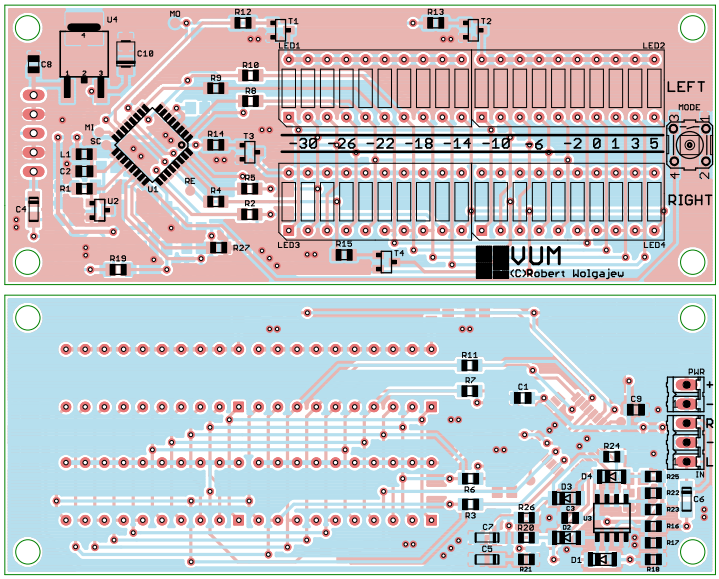
```
//Zmienna przechowująca domyślne (we Flash)
//wartości ADC dla kolejnych poziomów dBu: od –36dB do 5dB
const uint16_t dBuDefault[22] PROGMEM = {
    0, 15, 19, 24, 30, 38, 49, 60, 76, 97, 119,
    150, 188, 245, 321, 455, 638, 843, 933,
    999, 1021, 1024};
//Zmienna przechowująca wartości ADC
//dla kolejnych poziomów dBu zdefiniowanych przez użytkownika
uint16_t dBUser[22] EEMEM;
//Zmienna przechowująca wynikowe wartości ADC
//dla kolejnych poziomów dBu
uint16_t dB[22];
//Wskaźnik rodzaju skali: AA -> skala użytkownika
const uint8_t ScaleEE EEMEM;

//Funkcja odczytuje do tablicy dB[22] wynikowe wartości progowe
//w zależności od skali: predefiniowanej czy użytkownika

void initADC(void){
    //Sprawdzamy jakiego rodzaju skali należy używać:
    //predefiniowanej (dBu) czy użytkownika
    //mówi o tym zmienna EEPROM ScaleEE
    if(eeprom_read_byte(&ScaleEE) == 0xAA){
        for(uint8_t i = 0; i <22; i++)
            dB[i] = eeprom_read_word(&dBUser[i]);
    }else{
        for(uint8_t i = 0; i <22; i++)
            dB[i] = pgm_read_word(&dBuDefault[i]);
    }
}
```

VU. Zachowanie to możemy modyfikować poprzez zmianę wartości elementów integratora-detektora wartości szczytowej, a więc elementów R26, R27 i C7 (i odpowiednio R20, R21, C5 dla kanału lewego). Należy jednak pamiętać aby rezystor R26 (i odpowiednio R20) miał dużo mniejszą rezystancję niż R27 (i odpowiednio R21).

Wzmocnienie układu ustalają z kolei rezystory R25/R24 (i odpowiednio R18/R19) i tak jak wspomniano wcześniej, w naszym układzie wynosi ono około 2,47, a wynika z chęci dopasowania maksymalnej amplitudy wejściowego sygnału audio (+5 dBu, czyli ok. 3,9 Vp-p) do zakresu przetwarzania wbudowanego przetwornika ADC (4,096 V). Dla porządku, w **tabeli 1** umieściłem wartości poszczególnych poziomów sygnału audio i odpowiadające im obliczeniowe wartości napięć skutecznych (RMS), międzyszczytowych (peak-to-peak), amplitudy, wartości przetwarzania przetwornika ADC oraz rzeczywiste wartości pomiarowe z wyjścia detektora wartości szczytowej zmierzone multimetrem dla wejściowego sygnału sinusoidalnego o częstotliwości 1 kHz (i odpowiedniej amplitudzie).



Rysunek 3. Schemat płytki PCB wskaźnika wysterowania VUM

Zagadnienia programistyczne

Tyle w kwestii budowy urządzenia VUM. Przejdźmy zatem do kilku drobnych zagadnień programistycznych. Na początek mechanizm multipleksowania, którego prezentację rozpoczniemy od pokazania pliku nagłówkowego upraszczającego dostęp do zasobów mikrokontrolera i porządkującego kod aplikacji w tym zakresie. Mam świadomość, że nie jest to żadne „rocket science", 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.

Kod pliku nagłówkowego mechanizmu multipleksowania pokazano na **listingu 1**.

Jak widać, wprowadzono dwie zmienne globalne typu volatile: Led[2] i readyFor Update. Pierwsza z nich (tablica) przechowuje wartości (a w zasadzie wzór) przeznaczone do wyświetlenia na każdej z linijek LED (używanych jest 20 bitów w każdej z nich), zaś druga to flaga pozwalająca programowi głównemu na aktualizację właśnie tej zmiennej – Led[]. Potrzeba wprowadzenia drugiej zmiennej wynikała z konieczności synchronizacji chwili aktualizacji zmiennej Led[] z pracą funkcji multipleksującej linijki LED, tak by nie występowało zjawisko mieszania zawartości zmiennej Led[] wartościami z różnych pomiarów dokonywanych przecież w pętli głównej programu aplikacji, zwłaszcza że każdy element tej zmiennej jest 32-bitowy (zajmuje 4 bajty pamięci). Aktualizacja tej zmiennej następuje po pełnym cyklu multipleksu dla całego wyświetlacza LED. Dalej na **listingu 2** pokazano kod funkcji inicjującej mechanizm multipleksowania, zaś na **listingu 3** kod funkcji obsługi przerwania od Timer0, który odpowiada za mechanizm multipleksowania.

Prawda, że proste? Przejdźmy zatem do dwóch funkcji obsługi przetwornika ADC, których prezentację rozpoczniemy od funkcji inicjalizacyjnej, której wyłącznym zadaniem jest przygotowanie zmiennych używanych w docelowej funkcji pomiarowej. Ciało wspomnianej funkcji (wraz z definicją niezbędnych zmiennych) pokazano na **listingu 4**. Jak widać, funkcja powyższa korzysta z predefiniowanej (w pamięci Flash) tablicy dBuDefault[] przechowującej wartości przetwarzania przetwornika ADC dla kolejnych poziomów wejściowego sygnału audio (w skali dBu) lub podobnej tablicy dBUser[] (tyle że w pamięci EEPROM) przechowującej wartości przetwarzania przetwornika ADC dla kolejnych poziomów wejściowego sygnału audio zdefiniowanych przez użytkownika. Dzięki temu prostemu zabiegowi użytkownik urządzenia ma możliwość (poprzez zaprogramowanie wyłącznie pamięci EEPROM mikrokontrolera) zdefiniowania własnych wartości przetwornika ADC, dla których następuje zapalenie kolejnych diod LED linijki świetlnej.

Wybór konkretnej skali zdeterminowany jest wartością zmiennej ScaleEE umieszczonej pod adresem 0x00 we wspomnianej pamięci EEPROM. Jeśli jest ona równa 0xAA, zostanie wybrana skala użytkownika, w przeciwnym razie zastosowana zostanie skala domyślna dBu. W wyniku działania powyższej funkcji wartości skali wynikowej kopiowane są do zmiennej dB[] umieszczonej w pamięci RAM mikrokontrolera, co w założeniach ma przyspieszyć wykonywanie funkcji pomiarowej pokazanej później (gdyż odczyt z pamięci EEPROM jest co najmniej rząd wielkości wolniejszy niż analogiczny odczyt z pamięci RAM).

W przypadku skali użytkownika kolejne komórki pamięci EEPROM (począwszy od adresu 0x01 a skończywszy na 0x2C) powinny wtedy zawierać kolejne słowa tablicy dBUser[] (22 słowa 16-bitowe, bajt LSB jako pierwszy), przy czym pierwszy element tej tablicy (dBUser[0]) powinien zawierać wartość 0, zaś ostatni (dBUser[21]) wartość 1024. Wszystkie pośrednie powinny zawierać wartości przetwarzania przetwornika ADC (od najmniejszych do największych), przy których następuje zaświecenie kolejnej diody LED (1 do 20). Wartości te obliczamy według wzoru:

$$ADC = \frac{U}{4{,}096} \cdot 1024$$

gdzie

U – wartość napięcia [V] na wyjściu detektora wartości szczytowej.

Oczywiście w takim przypadku domyślna skala wydrukowana na płytce drukowanej nie ma wtedy zastosowania.

Przejdźmy zatem do implementacji funkcji odpowiedzialnej za wykonywanie pomiarów przy udziale wbudowanego przetwornika ADC, której ciało pokazano na **listingu 5**. W wyniku swojego działania funkcja zwraca wartość z zakresu 0…20 odpowiadającą wskazaniu (w dBu lub poziomach użytkownika) kolejnej diody LED na linijce świetlnej każdego z kanałów. Wartość ta jest przetwarzana w pętli głównej aplikacji na stosowny wzór (zmienna Led[]) przeznaczony do wyświetlenia na tejże linijce, zaś jego implementacja zależna jest od wybranego trybu wyświetlania. Za tę część

Listing 5. Kod funkcji odpowiedzialnej za wykonywanie pomiarów przy udziale wbudowanego przetwornika ADC

```
//Funkcja zwraca 0...20, czyli liczbę diod LED do zapalenia w słupku

uint8_t readADC(uint8_t Channel){
    uint16_t Voltage = 0;
    uint8_t Level;

    //Vref = AREF (default), internal Vref turned off
    ADMUX = Channel;

    //Wykonujemy 8 pomiarów by uśrednić wartość wynikową
    for(uint8_t i=0; i<8; i++){
        //Uruchomienie konwersji
        //Preskaler=8 (125kHz @ 1MHz)
        ADCSRA = (1<<ADEN)|(1<<ADSC)
            |(1<<ADPS1)|(1<<ADPS0);
        //Czekamy na zakończenie bieżącej konwersji – 120us
        while(ADCSRA & (1<<ADSC));
        Voltage +=  ADC;
    }
    Voltage /= 8;

    //Konwertujemy odczytaną wartość ADC do wartości wynikowej: 0...20
    for(Level = 0; Level <21; Level++)
        if(Voltage >= dB[Level] && Voltage < dB[Level+1]) break;

    return Level;
}
```

Listing 6. Fragment kodu obsługi aplikacji odpowiedzialny za przygotowanie zmiennej Led[]

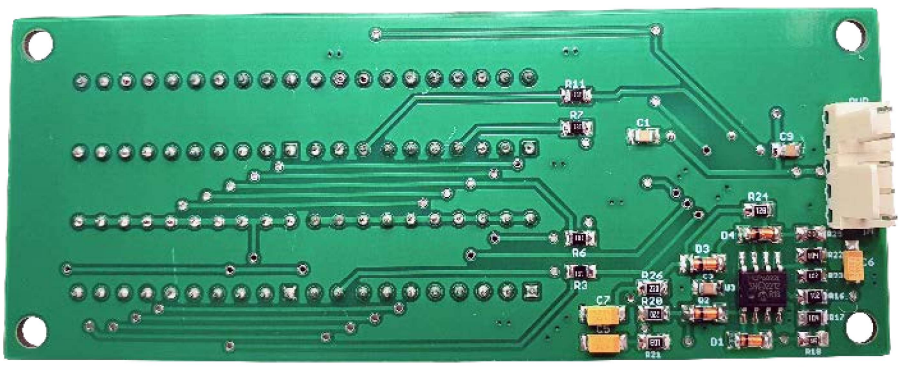
```
//Wykonujemy pomiary dla obu kanałów
Bar[0] = readADC(LEFT_CHANNEL);
Bar[1] = readADC(RIGHT_CHANNEL);

//Wartości obu pomiarów przeliczamy na wzór
//dla obu słupków LED, przy czym sposób przeliczenia
//zależy od bieżącego trybu pracy urządzenia:
//wyświetlanie pojedynczych pasków czy słupków
for(uint8_t i = 0; i < 2; i++){
    switch(Mode){
    case MODE_POINT:
    //Rysujemy pojedyńczą linię dla każdego wyświetlacza LED
        if(Bar[i] == 0)
            localLed[i] = 0;
        else localLed[i] = ((uint32_t) 1<<(Bar[i]-1));
    break;

    case MODE_BAR:
    //Rysujemy słupki dla każdej linijki LED
        localLed[i] = 0;
        for(uint8_t Val = 1; Val<21; Val++)
            if(Bar[i] >= Val)
                localLed[i] |= ((uint32_t) 1<<(Val-1));
            else break;
    break;

    case MODE_BAR_MAX:
    //Aktualizujemy maksima
        if(Bar[i] > maxBar[i]) maxBar[i] = Bar[i];
        //Co 200 ms zmniejszamy maksima o jeden pasek w dół
        if((++timer200ms%2) == 0) if(maxBar[i]) --maxBar[i];
        //Rysujemy słupki dla każdej linijki LED plus pasek maximum
        localLed[i] = 0;
        for(uint8_t Val = 1; Val<21; Val++)
            if(Bar[i] >= Val)
                localLed[i] |= ((uint32_t) 1<<(Val-1));
            else break;
        //Dorysowanie paska maximum
        if(maxBar[i])
            localLed[i] |= ((uint32_t) 1<<(maxBar[i]-1));
    break;
    }
}

//Czekamy na zezwolenie na aktualizację zmiennych Led[]
//dla funkcji ISR multipleksowania
while(readyForUpdate != 1);
ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
    readyForUpdate = 0;
    Led[0] = localLed[0];
    Led[1] = localLed[1];
}
```



Fotografia 1. Zmontowane urządzenie VUM od strony warstwy BOTTOM

programu obsługi aplikacji odpowiada kod pokazany na **listingu 6**. To byłoby tyle, jeśli chodzi o zagadnienia programistyczne, w związku z tym przejdźmy do szczegółów montażowych.

Montaż i uruchomienie

Schemat montażowy urządzenia VUM pokazano **rysunku 3**. Zaprojektowano niewielki, dwustronny obwód drukowany ze zdecydowaną przewagą elementów SMD montowanych po obu stronach laminatu (dla zmniejszenia wymiarów PCB). Montaż urządzenia rozpoczynamy od warstwy TOP, gdzie w pierwszej kolejności montujemy mikrokontroler sterujący. Najprostszym sposobem montażu elementu o takim zagęszczeniu wyprowadzeń, niewymagającym jednocześnie posiadania specjalistycznego sprzętu, jest użycie typowej stacji lutowniczej, dobrej jakości cyny z odpowiednią ilością topnika oraz dość cienkiej plecionki rozlutowniczej, która umożliwi usunięcie nadmiaru cyny spomiędzy wyprowadzeń złącza. Należy przy tym uważać, by nie uszkodzić termicznie tegoż elementu. Jakość tak wykonanego połączenia sprawdzamy pod lupą, korzystając z najprostszego miernika pozwalającego sprawdzić ciągłość połączeń. Wspomniana kontrola będzie znacznie łatwiejsza, jeśli zmontowaną płytkę przemyjemy alkoholem izopropylowym w celu wypłukania nadmiaru kalafonii lutowniczej.

W kolejnym kroku montujemy pozostałe półprzewodniki, a na końcu elementy bierne. W tym momencie przechodzimy na warstwę BOTTOM, gdzie podobnie jak poprzednio, w pierwszej kolejności przylutowujemy półprzewodniki, zaś na końcu elementy bierne. Dalej, wracamy na warstwę TOP, gdzie przylutowujemy linijki LED oraz microswitch MODE. Ostatnim etapem montażu jest przylutowanie gniazd połączeniowych PWR i IN, co możemy wykonać zarówno od strony BOTTOM (preferowane z uwagi na późniejsze podłączenia) lub TOP w zależności od docelowego sposobu montażu urządzenia.

Poprawnie zmontowany układ nie wymaga żadnego specjalnego procesu uruchamiania i powinien działać tuż po podłączeniu zasilania. W przypadku monitorowania sygnałów audio o amplitudach przekraczających zakres napięć wejściowych urządzenia należy na wejściu zastosować prosty dzielnik napięcia w postaci potencjometru o rezystancji w granicach 50 kΩ lub też zmienić wzmocnienie wejściowego układu audio zbudowanego ze wzmacniaczy operacyjnych. Na **fotografii 1** pokazano zmontowane urządzenie VUM od strony warstwy BOTTOM, zaś to samo urządzenie od strony warstwy TOP na **fotografii tytułowej**.

Robert Wołgajew, EP