W ofercie AVT*
AVT6040

Najważniejsze parametry:
- dopuszczalny zakres mierzonych pojemności: 0,1 pF...999 µF,
- zakresy pomiarowe: 100 pF, 1000 pF, 100 nF, 1000 nF, 100 µF, 1000 µF,
- dokładność pomiarów: 2%,
- maksymalna częstotliwość odczytów: 2 Hz,
- czas pracy na baterii AAA(1): 6 miesięcy.

(1) Przy zachowaniu założonych warunków użytkowania (szczegóły w tekście artykułu).

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

AVT5570	Przyrząd do formowania kondensatorów elektrolitycznych (EP 1/2017)
AVT614	Warsztatowy tester kondensatorów (EdW 10/2004)
AVT5003	Tester elementów elektronicznych (EP 3/2001)
AVT2404	Miernik rezystancji kondensatorów (EdW 3/2001)

* **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**.

cMeter – przenośny miernik pojemności

W praktyce każdego elektronika amatora czy profesjonalisty nieodzownym elementem codziennej rutyny są wszelkiego rodzaju naprawy, a te – jak łatwo się domyślić – wiążą się z nieustającą koniecznością testowania „podejrzanych" elementów. Jednymi z najczęściej spotykanych komponentów w sprzęcie elektronicznym są rezystory i kondensatory, przy czym o ile dość łatwym zadaniem okazuje się organoleptyczne zdiagnozowanie uszkodzenia rezystora, to już nie tak oczywiste jest postawienie poprawnej diagnozy uszkodzenia kondensatora (li tylko na podstawie wyglądu „osobnika").

Co prawda większość współczesnych multimetrów wyposażonych jest w funkcjonalność pomiaru pojemności, lecz nie wszystkie realizują ją dobrze (jeśli weźmiemy pod uwagę zarówno dokładność pomiaru, jak i funkcjonalność). Użycie przyrządu wielofunkcyjnego w tym celu wyklucza ponadto jego zastosowanie do innych zadań w tym samym czasie (co z kolei okazuje się często pożądane w praktyce). Właśnie tego typu dylematy stały się przyczynkiem do opracowania niniejszego projektu miniaturowego, przenośnego miernika pojemności o nazwie cMeter. Przyznać muszę, że na początku prac konstrukcyjnych skupiłem się na poszukiwaniu optymalnej metody pomiaru pojemności, zakładając dość duży (w gruncie rzeczy wręcz ekstremalny) zakres wartości pojemności badanych elementów. Jak się wkrótce okazało – najlepszą (a może najprostszą?) metodą jest książkowy pomiar czasu ładowania kondensatora mierzonego, który to czas jest proporcjonalny do pojemności kondensatora oraz do rezystancji szeregowej obwodu pomiarowego. Dość szybko przypomniałem sobie, że dobrym źródłem wiedzy praktycznej z wielu tematów z pogranicza elektroniki i programowania jest strona http://elm-chan.org/, czyli serwis dobrze znanego twórcy pakietów FatFS i PetitFS. Dokładnie tak było i tym razem. To właśnie we wspomnianym serwisie znalazłem ciekawą, choć bardzo prostą implementację

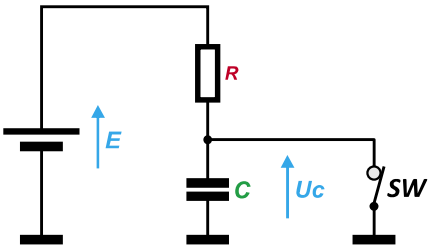
Wykaz elementów:

Rezystory:
R1: 976 kΩ 1% (SMD 0603)
R2: 562 kΩ 1% (SMD 0603)
R3: 10 kΩ (SMD 0603)
R4: 10 kΩ 1% (metalizowany, mini-MELF 0204)
R5: 100 Ω 1% (metalizowany, mini-MELF 0204)
R6, R7: 39 kΩ 1% (metalizowany, mini-MELF 0204)
R8: 680 Ω 1% (metalizowany, mini-MELF 0204)
R9: 3,3 MΩ 1% (metalizowany, mini-MELF 0204)
R10...R17: 330 Ω (SMD 0603)

Kondensatory:
C1, C2: ceramiczny X7R 100 nF (SMD 0603)
C3, C4: ceramiczny X7R 10 µF (SMD 0603)

Półprzewodniki:
U1: ATtiny44 (SO14)
U2: MCP1640T-I/CHY (SOT23-6)
U3: M74HC4094 (SO16)
LED: wyświetlacz 7-segmentowy LED typu OSK4039A-IG lub podobny w wybranym kolorze
T1: BC807 (SOT23)
T2...T5: MMBTRA105SS (SOT23)

Pozostałe:
L1: dławik drutowy SMD 4,7 µH typu WLPN303015M4R7PB (SMD 3×3 mm)
PWR: mikroprzełącznik TACT SMD typu TACTM-67N-F NINIGI (wysokość 7...9 mm)
BATT: koszyczek baterii AAA typu 1021 KEYSTONE



Rysunek 1. Typowy, szeregowy obwód RC, który odpowiada za ładowanie kondensatora

wspomnianego wcześniej mechanizmu pomiaru pojemności – i to właśnie ona stała się inspiracją do stworzenia niniejszego projektu. Zacznijmy jednak od podstaw. Na **rysunku 1** pokazano typowy, szeregowy obwód RC, odpowiadający za ładowanie kondensatora.

W obwodzie z rysunku 1 otwarcie przełącznika SW rozpoczyna proces ładowania kondensatora, podczas którego napięcie na zaciskach kondensatora wyraża się następującym wzorem:

$$U_c = E \cdot (1 - e^{\frac{-t}{R \cdot C}})$$

gdzie
U_c – napięcie na zaciskach kondensatora [V],
E – napięcie idealnego źródła zasilania [V],
e – liczba Eulera (zwana również liczbą Nepera),
t – czas ładowania kondensatora [s],
R – rezystancja rezystora szeregowego [Ω],
C – pojemność kondensatora [F].

Jak widać, jeśli przyjmiemy stałe wartości E oraz R, napięcie na zaciskach mierzonego kondensatora zależne będzie **wyłącznie** od jego pojemności (C) oraz czasu ładowania (t). Z powyższego wzoru możemy wyznaczyć czas ładowania dla konkretnego napięcia U_c, który wyraża się następującym wzorem:

$$t = R \cdot C \cdot ln(1 - \frac{U_c}{E})$$

Wyrażając Uc jako ułamek E (ułamek napięcia zasilania), czyli przyjmując

$$U_c = THRESH \cdot E$$

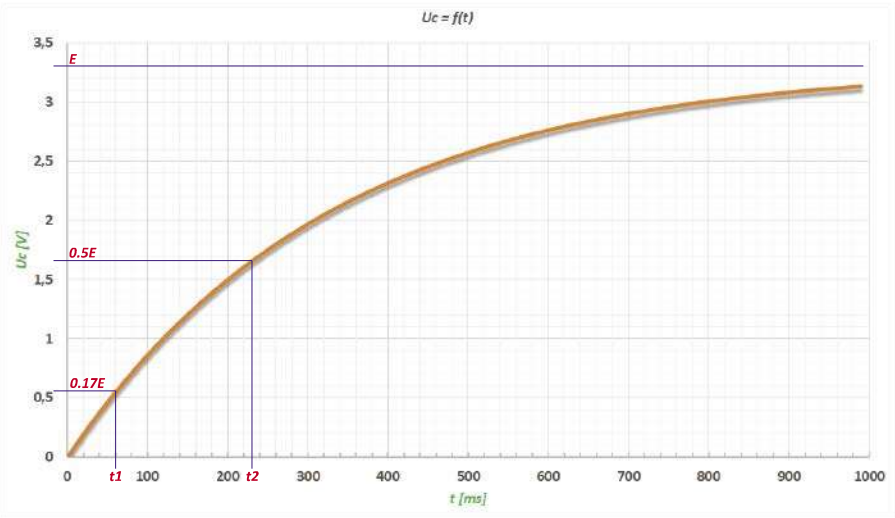
otrzymujemy:

$$t = -R \cdot C \cdot ln(1 - THRESH)$$

Przekształcając powyższy wzór w celu otrzymania pojemności, dochodzimy do poniższego wyrażenia:

$$C = \frac{-t}{R \cdot ln(1 - THRESH)}$$

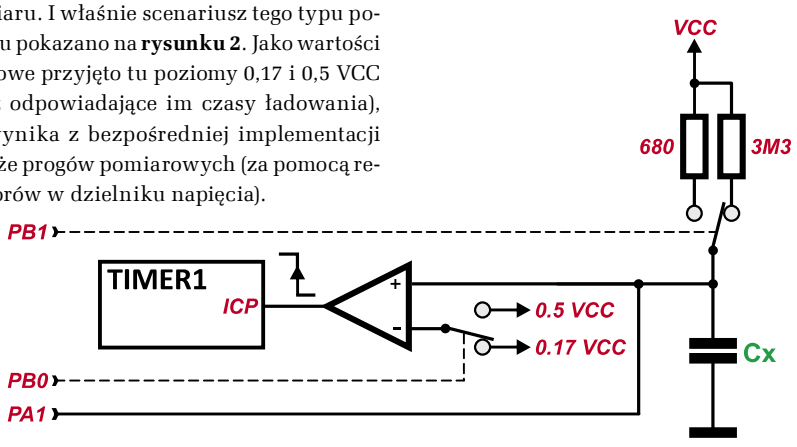
Niezbędna do ustalenia pojemność kondensatora mierzonego wynika zatem wyłącznie ze znanej wartości rezystora szeregowego, napięcia progowego (również znanego) i czasu ładowania (łatwego do zmierzenia), czyli – zakładając stałe wartości R i *THRESH* – proporcjonalna jest **wyłącznie** do czasu ładowania. I to jest clou mechanizmu pomiarowego – które pozornie mogłoby wydawać się banalne, ale nie wyczerpuje jeszcze treści naszych rozważań. Proponowane powyżej



Rysunek 2. Wykres ładowania kondensatora

mechanizmy zakładają bowiem, iż początkowe napięcie na zaciskach kondensatora (którego pojemność chcemy zmierzyć) równe jest zero. Takie zjawisko jednak **NIGDY** nie występuje w rzeczywistych układach, gdyż po pierwsze – każdy rzeczywisty kondensator ma pewną rezystancję szeregową, która „spowalnia" proces jego rozładowania, a po drugie – rozładowanie takiego kondensatora do wspomnianego zera lub naładowanie go do wartości napięcia zasilania wymagałoby sporo czasu. Oczekiwanie z kolei byłoby niepożądane w trakcie procesu pomiarowego, gdyż – jak łatwo się domyślić – chcemy mierzoną wartość otrzymać niemalże natychmiast a nie np. po 10 sekundach, prawda? Tak naprawdę zresztą, kto gwarantuje nam, że mierzony kondensator został rozładowany do ZERA? Jak sobie poradzić z tym problemem? Dość łatwo! Należy zmierzyć czasy ładowania mierzonego kondensatora dla pewnych dwóch wartości progowych napięcia (*THRESH*) i na podstawie różnicy tych czasów obliczyć wartość mierzonej pojemności. Same wartości progowe (*THRESH*) nie są tutaj krytyczne, lecz powinny być od siebie na tyle „oddalone", by, po pierwsze, zapewnić odpowiednią dokładność pomiaru, a po drugie – maksymalnie skrócić czas tegoż pomiaru. I właśnie scenariusz tego typu pomiaru pokazano na **rysunku 2**. Jako wartości progowe przyjęto tu poziomy 0,17 i 0,5 VCC (oraz odpowiadające im czasy ładowania), co wynika z bezpośredniej implementacji tychże progów pomiarowych (za pomocą rezystorów w dzielniku napięcia).

A skoro mowa o wartościach *progowych*… czy coś Wam „świta" w głowach? Najłatwiej jest użyć komparatora analogowego z odpowiednio ustawionym napięciem odniesienia (V_{REF}), by stosowny pomiar mógł być wykonany w najprostszy sposób. A jeśli mowa o komparatorze analogowym, to nie sposób nie użyć tego wbudowanego w mikrokontroler, zwłaszcza że dość często potrafi on wyzwalać zdarzenie przechwycenia pracującego licznika (Timer1), a co za tym idzie – może w prosty sposób „generować" znaczniki czasowe. Tak, właśnie tym tropem pójdziemy! Ale istnieje jeszcze jeden aspekt omawianego mechanizmu, nie mniej istotny. Skoro chcemy mierzyć pojemność kondensatorów o dość dużym zakresie (powiedzmy – od pojedynczych pF do setek µF), to czas pomiaru dla tych największych (przy ustalonej wartości rezystancji szeregowej R) może okazać się bardzo długi (rzędu dziesiątek sekund), czego z pewnością byśmy nie chcieli. Jak rozwiązać ten problem? Znowu, w najprostszy sposób. Należy zmieniać wartość rezystancji szeregowej (czyli de facto zakres czasów ładowania kondensatora), tak by nawet dla „dużych" kondensatorów czas ten był na akceptowalnym poziomie. Wniosek z naszych rozważań jest taki, że musimy mieć możliwość



Rysunek 3. Funkcjonalna wizualizacja pełnego mechanizmu pomiarowego urządzenia cMeter

po pierwsze – przełączania wartości rezystora szeregowego, a po drugie – zmiany napięcia referencyjnego, aby móc dokonać pomiaru wspomnianej pojemności, przy zachowaniu odpowiedniej dokładności tychże pomiarów. Funkcjonalną wizualizację pełnego mechanizmu pomiarowego, o którym napisałem powyżej, pokazano na **rysunku 3**.

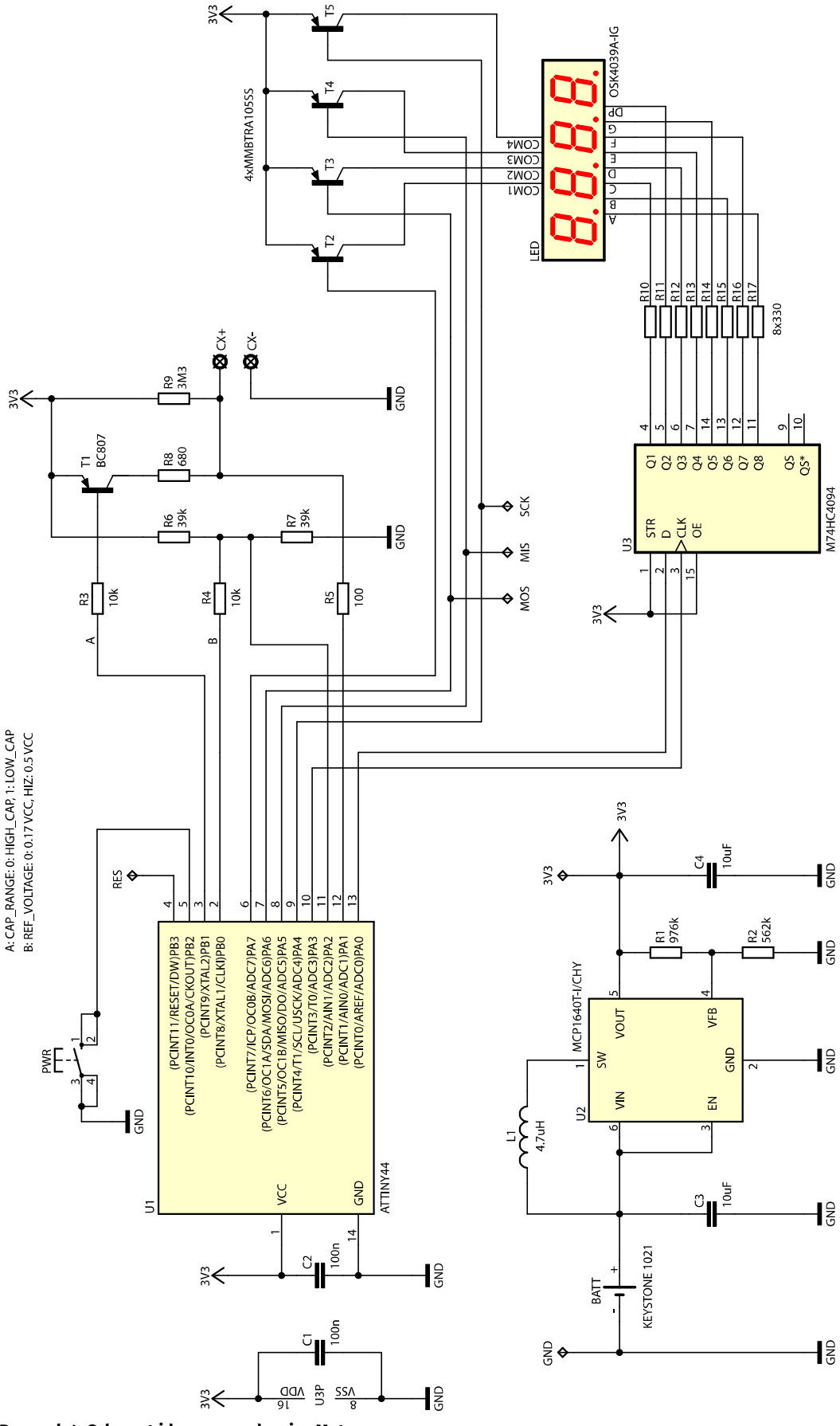
I właśnie na bazie powyższych założeń powstał projekt urządzenia cMeter, którego schemat ideowy znalazł się na **rysunku 4**.

Jak widać, zaprojektowano bardzo prosty system mikroprocesorowy. Jego serce stanowi niewielki mikrokontroler ATtiny44 firmy Microchip (dawniej Atmel), taktowany wewnętrznym oscylatorem RC o częstotliwości 8 MHz i realizujący całą założoną funkcjonalność urządzenia. Mikrokontroler nasz realizuje następujące zadania:

- steruje pracą szeregowego rejestru przesuwnego 74HC4094 (wyprowadzenia PA3→Clock, PA0→Data), za pomocą którego obsługuje 7-segmentowy wyświetlacz LED o organizacji 4 znaków w konfiguracji wspólnej anody (wyprowadzenia PA7…PA4 mikrokontrolera),
- steruje zewnętrznymi obwodami pomiarowymi (wyprowadzenie PB1 jest odpowiedzialne za wybór wartości rezystora szeregowego [3,3 MΩ lub 680 Ω], zaś PB0 umożliwia wybór napięcia referencyjnego [0,17 VCC lub 0,5 VCC]),
- steruje pracą wbudowanego komparatora analogowego, stanowiącego podstawę mechanizmu pomiarowego (wyprowadzenia PA2, PA1), inicjując tym samym proces rozładowania i następującego po nim ładowania kondensatora mierzonego (PA1=1/0).

Warto już w tym momencie podkreślić, iż wszystkie rezystory w torze pomiarowym przewidziano jako rezystory metalizowane w obudowach mini-MELF 0204, by do minimum zredukować ewentualne szumy wprowadzane przez typowe i tanie elementy cienko– i grubowarstwowe (thin film, thick film).

Wybór mikrokontrolera ATtiny44 oraz podłączonego do jego wyprowadzeń rejestru



Rysunek 4. Schemat ideowy urządzenia cMeter

szeregowego 74HC4094 mógłby się wydawać dość wątpliwy, jeśli bierzemy pod uwagę, że bez problemu można byłoby wybrać mikrokontroler o odpowiedniej liczbie portów wyjściowych, zamiast stosować przedmiotowy mikrokontroler i dodatkowy rejestr przesuwny. To jednak tylko pozory! Nie chciałem bowiem stosować mikrokontrolera o większej liczbie (niepotrzebnych) wyprowadzeń, a co za tym idzie – o niewygodnej do lutowania dla amatora obudowie (TQFP32). Istnieje jednak drugi, ważniejszy powód: zastosowanie rejestru przesuwnego 74HC4094 zdecydowanie upraszczało projekt obwodu drukowanego, a układ tego rodzaju kosztuje około 1,50 zł. Wspomniane wcześniej wspólne anody wyświetlacza LED sterowane są poprzez proste klucze tranzystorowe T2… T5 z uwagi na dość duże prądy o wartościach rzędu 32 mA (8×4 mA na segment). Z kolei katody naszych elementów LED obsługiwane są przez wyprowadzenia rejestru przesuwnego i, jak już można się domyślić, do ich obsługi (podobnie jak wspólnych anod) zastosowano doskonale znany mechanizm multipleksowania. Jest to typowe rozwiązanie problemu tego typu, a polega na sekwencyjnym sterowaniu kolejnych cyfr wyświetlacza LED. Wspomniana procedura, wykonywana dostatecznie szybko (w naszym wypadku 60 razy na sekundę dla każdej wspólnej anody), pozwala na obsłużenie 32 segmentów wyświetlacza LED przy udziale wyłącznie 6 wyprowadzeń mikrokontrolera. Już teraz nadmienię, że użyjemy w tym celu układu czasowo-licznikowego Timer0 wbudowanego w strukturę mikrokontrolera, który pracował będzie w trybie CTC i wywoływał stosowne przerwanie systemowe (od porównania) 240 razy na sekundę (czyli 60 razy dla każdej wspólnej anody), obsługując właściwy mechanizm multipleksowania. Wróćmy jednak do schematu ideowego naszego urządzenia, gdyż kilku niezbędnych słów wyjaśnienia wymaga blok zasilający. Zdecydowałem się na zastosowanie popularnej baterii typu AAA o pojemności w granicach 1300 mAh (w przypadku dobrych baterii alkalicznych lub litowych), a wybór ten podyktowany był łatwą dostępnością ogniw i ich niską ceną. Z uwagi na to konieczne stało się użycie prostej, acz nowoczesnej przetwornicy step-up pod postacią układu scalonego MCP1640 firmy Microchip, która dostarcza napięcie wyjściowe rzędu 3,3 V (regulowane dzielnikiem rezystancyjnym) już przy napięciu wejściowym na poziomie 0,65 V i zapewnia niski prąd spoczynkowy w granicach 1 µA oraz wysoką sprawność, co nie jest bez znaczenia dla typu zastosowanego źródła napięcia zasilającego. Aby ocenić, jak długo urządzenie cMeter pracować będzie na pojedynczej baterii AAA, należy zastanowić się, z jakich etapów składa się cykl jego pracy i jakiej wielkości prądy pobiera wtedy ze źródła zasilania. Przystępując do obliczeń, przyjąłem następujący podział cyklu pracy urządzenia:

- czas trybu Power Down (uśpienia), który trwa z dużym przybliżeniem 24 h/dobę i podczas którego pobierany jest prąd rzędu 200 µA. Dodajmy, że „większość" tego prądu to prąd spoczynkowy przetwornicy, który w przypadku dużej różnicy między napięciem wejściowym a wyjściowym, co ma miejsce w naszym przykładzie, znacznie przekracza (o rząd wielkości) deklarowany przez producenta, minimalny prąd spoczynkowy oraz prąd pobierany przez obwód pomiarowy – rezystorowy dzielnik napięcia przyłączony na stałe pomiędzy bieguny zasilania.
- czas wykonywania i wyświetlania pomiarów, który zakładamy na poziomie 30 s i podczas którego pobierany jest średni prąd na poziomie 20 mA.

Założono ponadto, iż wybudzanie urządzenia i następujący po nim pomiar ma miejsce 20 razy na dobę, co oznacza, że urządzenie używane jest maksymalnie 10 minut dziennie. Przy założeniach jak wyżej otrzymano teoretycznie, niespełna 6 miesięcy pracy na pojedynczej baterii AAA, co wydaje się wartością co najmniej satysfakcjonującą. I na koniec, wyłącznie dla porządku dodam,

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

```
//Definicje portów rejestru przesuwnego
#define SERIAL_PORT PORTA
#define SERIAL_DDR DDRA
#define SERIAL_DAT_NR PA0
#define SERIAL_CLK_NR PA3

//Porty rejestru przesuwnego (DAT, CLK), jako wyjściowe ze stanem 0
#define SERIAL_PORT_AS_OUTPUT SERIAL_DDR |= (1<<SERIAL_DAT_NR)|(1<<SERIAL_CLK_NR)

#define SERIAL_DAT_SET SERIAL_PORT |= (1<<SERIAL_DAT_NR)
#define SERIAL_DAT_RESET SERIAL_PORT &= ~(1<<SERIAL_DAT_NR)
#define SERIAL_CLK_SET SERIAL_PORT |= (1<<SERIAL_CLK_NR)
#define SERIAL_CLK_RESET SERIAL_PORT &= ~(1<<SERIAL_CLK_NR)
#define SERIAL_CLK_TICK SERIAL_CLK_SET; SERIAL_CLK_RESET

//Definicje konfiguracji poszczególnych segmentów (katod) rejestru przesuwnego
#define SEG_A (1<<7)
#define SEG_B (1<<5)
#define SEG_C (1<<0)
#define SEG_D (1<<2)
#define SEG_E (1<<3)
#define SEG_F (1<<6)
#define SEG_G (1<<4)
#define SEG_DP (1<<1)

//Definicje portu wspólnych anod – tranzystory sterujące
#define COM_PORT PORTA
#define COM_DDR DDRA

//Definicje konfiguracji poszczególnych wspólnych anod
#define COM_DIG1 PA7
#define COM_DIG2 PA6
#define COM_DIG3 PA5
#define COM_DIG4 PA4

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

//Definicje dla Timera0 odpowiedzialnego za multipleksowanie wyświetlacza LED
#define START_MUX_TIMER TCCR0B = (1<<CS02)      //Preskaler = 256
#define STOP_MUX_TIMER TCCR0B = 0

//Definicja bitu odpowiedzialnego za kropkę dziesiętną
#define DP_BIT 0b10000000

//Indexy znaków specjalnych
#define CHAR_space 10
#define CHAR_p 11
#define CHAR_n 12
#define CHAR_u 13
#define CHAR_b 14
#define CHAR_A 15
#define CHAR_t 16
#define CHAR_h 17
#define CHAR_i 18
#define CHAR_g 19
#define CHAR_C 20
#define CHAR_L 21
#define CHAR_r 22
#define CHAR_o 23
#define CHAR_d 24
#define CHAR_E 25
#define CHAR_F 26

//Deklaracje zmiennych globalnych
extern volatile uint8_t Digit[4];              //Zmienna przechowująca wartość wyświetlaną na wyświetlaczu LED
extern volatile uint8_t readyForUpdate;        //Zezwolenie na atomową zmianę zmiennych
```

Listing 1. Plik nagłówkowy mechanizmu multipleksowania

że mikrokontroler nasz korzysta również z wbudowanego w swoją strukturę przetwornika analogowo-cyfrowego, dzięki któremu dokonuje ustawicznego pomiaru napięcia zasilania systemu mikroprocesorowego i w przypadku jego nieodpowiedniej wartości podejmuje stosowne czynności. Tyle w kwestiach funkcjonalnych – przejdźmy zatem do zagadnień implementacyjnych.

Mam świadomość, że prezentowana koncepcja nie stanowi być może *rocket science* ani rozwiązania na wskroś uniwersalnego, jednak chciałem pokazać Wam, jak w efektywny i efektowny sposób ogarnąć tego rodzaju zagadnienie programistyczne, czyniąc sam proces programowania niezmiernie przyjemnym. Nieskromnie powiem, że w moim przekonaniu właśnie w ten przejrzysty sposób powinno się konstruować moduły obsługi danych peryferiów, gdyż jakakolwiek modyfikacja sprowadza się wtedy do kosmetycznych i prostych do wykonania zmian. Na początek przeanalizujmy plik nagłówkowy mechanizmu multipleksowania, który pokazano na **listingu 1**, a dzięki któremu porządkujemy późniejszy kod źródłowy, czyniąc go bardzo czytelnym i jednocześnie upraszczając proces wprowadzania zmian. Plik ten zarówno definiuje główne ustawienia sprzętowe, jak i wprowadza niezbędne zmienne – zadeklarowano w nim szereg zmiennych globalnych (typu volatile z uwagi na ich użycie tak w programie głównym, jak i funkcji ISR), przechowujących ustawienia poszczególnych elementów wyświetlacza LED. Już na tym etapie musimy zdefiniować kilka stałych opisujących wzorce znaków czy też upraszczających dostęp do portów procesora, gdyż zależy nam, by nasza procedura obsługi przerwania multipleksująca wyświetlacz była jak najkrótsza. Definicje, o których mowa, pokazano na **listingu 2**.

Jak widać, wspomniane definicje zostały umieszczone w pamięci RAM mikrokontrolera. Jest to pewnego rodzaju „marnotrawstwo", gdyż stałe te z powodzeniem można (a nawet wypada) umieścić w pamięci Flash mikrokontrolera, aby nie marnować cennej pamięci RAM (zwłaszcza że wartości tych stałych nasz kompilator i tak musi umieścić, a następnie odczytać z tejże pamięci Flash na starcie działania programu). Dokładnie tak postępowałem dotychczas, pisząc oprogramowanie embedded – pamiętajmy jednak, że dostęp do pamięci Flash jest nieco wolniejszy niż odczyt stałych z pamięci RAM (dokładnie 5 taktów zegara zamiast 2). W związku z tym zdecydowałem się na powyższe rozwiązanie, zwłaszcza że wykorzystanie pamięci RAM w naszej aplikacji utrzymuje się na poziomie 20%. Wartość ta stanowi pozornie niewielki przyrost szybkości, ale zawsze coś. Skądinąd idea taka jest zgodna z podejściem twórców Androida, charakteryzowanym przez frazę: „dlaczego nieużywana pamięć RAM ma leżeć odłogiem"? Abstrahując już od celowości i sensowności takiego postępowania, brnijmy dalej we wskazanym kierunku. Pora na zaprezentowanie funkcji konfigurującej zarówno mechanizm multipleksowania, jak i niezbędne ustawienia sprzętowe, której ciało znajduje się na **listingu 3**.

Dalej, na **listingu 4** uwidoczniono funkcję obsługi przerwania od porównania wartości licznika Timer0 z rejestrem OCR0A,

```
//Definicje znaków wyświetlacza LED (aktywny stan "0", gdyż sterujemy katodami diod LED)
const uint8_t digPattern[] =
{
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_F),      //0
    (uint8_t) ~(SEG_B|SEG_C),                              //1
    (uint8_t) ~(SEG_A|SEG_B|SEG_D|SEG_E|SEG_G),            //2
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_G),            //3
    (uint8_t) ~(SEG_B|SEG_C|SEG_F|SEG_G),                  //4
    (uint8_t) ~(SEG_A|SEG_C|SEG_D|SEG_F|SEG_G),            //5
    (uint8_t) ~(SEG_A|SEG_C|SEG_D|SEG_E|SEG_F|SEG_G),      //6
    (uint8_t) ~(SEG_A|SEG_B|SEG_C),  //7
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_F|SEG_G),//8
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_F|SEG_G),      //9
    0xFF,                                                  //Wyświetlacz wygaszony (10)
    (uint8_t) ~(SEG_A|SEG_B|SEG_G|SEG_F|SEG_E),            //p - 11
    (uint8_t) ~(SEG_E|SEG_G|SEG_C),                        //n - 12
    (uint8_t) ~(SEG_C|SEG_D|SEG_E),                        //u - 13
    (uint8_t) ~(SEG_G|SEG_F|SEG_E|SEG_D|SEG_C),            //b - 14
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_E|SEG_F|SEG_G),      //A - 15
    (uint8_t) ~(SEG_C|SEG_D|SEG_E|SEG_F|SEG_G),            //t - 16
    (uint8_t) ~(SEG_C|SEG_E|SEG_F|SEG_G),                  //h - 17
    (uint8_t) ~(SEG_E),                                    //i - 18
    (uint8_t) ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_F|SEG_G),      //g - 19
    (uint8_t) ~(SEG_A|SEG_D|SEG_E|SEG_F),                  //C - 20
    (uint8_t) ~(SEG_D|SEG_E|SEG_F),                        //L - 21
    (uint8_t) ~(SEG_E|SEG_G),                              //r - 22
    (uint8_t) ~(SEG_C|SEG_D|SEG_E|SEG_G),                  //o - 23
    (uint8_t) ~(SEG_B|SEG_C|SEG_D|SEG_E|SEG_G),            //d - 24
    (uint8_t) ~(SEG_A|SEG_D|SEG_E|SEG_F|SEG_G),            //E - 25
    (uint8_t) ~(SEG_A|SEG_E|SEG_F|SEG_G)                   //F - 26
};

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

Listing 2. Definicje niezbędnych stałych mechanizmu multipleksowania

```
void initMultiplex(void)
{
    //Porty wspólnych anod i interfejsu szeregowego jako wyjściowe ze stanami
    //nieaktywnymi na wyjściach
    SERIAL_PORT_AS_OUTPUT;
    COM_BLANK;
    COM_AS_OUTPUT;

    //Konfiguracja Timera0 w celu generowania przerwania do obsługi multipleksowania
    //wyświetlacza LED (240 Hz)
    TCCR0A = (1<<WGM01);                    //Tryb CTC
    OCR0A = 129;                            //240 Hz (ISR co 4.17 ms)
    START_MUX_TIMER;                        //Uruchomienie Timera0
    TIMSK0 = (1<<OCIE0A);                   //Uruchomienie przerwania Timer0 Output
                                            //Compare Match A
}
```

Listing 3. Funkcja konfigurująca mechanizm multipleksowania

```
ISR(TIM0_COMPA_vect)
{
    static uint8_t Nr;                                  //Numer kolejnej cyfry przeznaczonej do wyświetlenia
    uint8_t Dig = Digit[Nr];                            //Optymalizacja volatile


    //Wyłączenie wspólnych anod wyświetlaczy LED
    COM_BLANK;
    //Wystawienie wzoru na port katod (rejestr przesuwny) uwzględniające bit kropki dziesiętnej (bit 7 -> DP_BIT)
    if(Dig & DP_BIT) serialSend(digPattern[Dig & (~DP_BIT)] & (~SEG_DP)); else serialSend(digPattern[Dig]);
    //Włączenie odpowiedniej wspólnej anody (aktywny stan "0")
    COM_PORT &= comPattern[Nr];
    //Wybranie kolejnej wspólnej anody
    Nr = (Nr+1) & 0x03;
    //Zezwolenie na atomową zmianę zmienej Digit[] w funkcji Main
    if(Nr == 0) readyForUpdate = 1; else readyForUpdate = 0;
}
```

Listing 4. Funkcja obsługi przerwania realizująca mechanizm multipleksowania

```
inline void serialSend(uint8_t Byte)
{
    //Wysyłamy bajt do rejestru przesuwnego (zbocze rosnące na CLK) przesuwając kolejne
    //bity na wyjście DAT począwszy od bitu MSB
    if(Byte & 0x80) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x40) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x20) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x10) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x08) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x04) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x02) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
    if(Byte & 0x01) SERIAL_DAT_SET; else SERIAL_DAT_RESET; SERIAL_CLK_TICK;
}
```

Listing 5. Funkcja odpowiedzialna za przesłanie bajtu danych do rejestru przesuwnego

odpowiedzialną za realizację mechanizmu multipleksowania wyświetlacza. W implementacji tej wzięto pod uwagę obsługę kropki dziesiętnej wyświetlacza LED.

I na sam koniec, na **listingu 5**, pokazano funkcję odpowiedzialną za przesłanie bajtu danych do rejestru przesuwnego.

Warto przez chwilę zastanowić się nad znaczeniem nieopisanej wcześniej zmiennej readyForUpdate. Jest to zmienna, która funkcji głównej aplikacji użytkownika wskazuje

```
//Ustawienia portów odpowiedzialnych za proces pomiaru pojemności
#define MEAS_PORT PORTB
#define MEAS_DDR DDRB
#define MEAS_CAP_RNG_NR PB1
#define MEAS_REF_VLG_NR PB0

#define MEAS_SELECT_CAP_AS_OUTPUT MEAS_DDR |= (1<<MEAS_CAP_RNG_NR)
#define MEAS_SELECT_LOW_CAP MEAS_PORT |= (1<<MEAS_CAP_RNG_NR)
#define MEAS_SELECT_HIGH_CAP MEAS_PORT &= ~(1<<MEAS_CAP_RNG_NR)
#define MEAS_LOW_CAP_IS_SELECTED (MEAS_PORT & (1<<MEAS_CAP_RNG_NR))

#define MEAS_SELECT_LOW_REF MEAS_DDR |= (1<<MEAS_REF_VLG_NR)          //Port PB0, jako wyjściowy ze stanem 0 (0.17 VCC)
#define MEAS_SELECT_HIGH_REF MEAS_DDR &= ~(1<<MEAS_REF_VLG_NR)        //Port PB0, jako HiZ (0.5 VCC)
#define MEAS_LOW_REF_IS_SELECTED (MEAS_DDR & (1<<MEAS_REF_VLG_NR))
#define MEAS_REF_PULL_UP_ON MEAS_PORT |= (1<<MEAS_REF_VLG_NR)         //Podciągnięcie portu referencji pod VCC (w Power Down)
#define MEAS_REF_PULL_UP_OFF MEAS_PORT &= ~(1<<MEAS_REF_VLG_NR)

//Port rozładowania mierzonego kondensatora (tym samym port komparatora AIN0)
#define DISCHARGE_DDR DDRA
#define DISCHARGE_PORT PORTA
#define DISCHARGE_NR PA1
#define MEAS_DISCHARGE_CAP DISCHARGE_DDR |= (1<<DISCHARGE_NR)          //Port PA1, jako wyjściowy ze stanem 0
                                                                       //(rozładowanie kondensatora mierzonego)
#define MEAS_CHARGE_CAP DISCHARGE_DDR &= ~(1<<DISCHARGE_NR)            //Port PA1, jako HiZ (ładowanie kondensatora mierzonego)
#define MEAS_DISCHARGE_PULL_UP_ON DISCHARGE_PORT |= (1<<DISCHARGE_NR)  //Podciągnięcie portu rozładowania pod VCC (w Power Down)
#define MEAS_DISCHARGE_PULL_UP_OFF DISCHARGE_PORT &= ~(1<<DISCHARGE_NR)

//Ustawienia Timera1 odpowiedzialnego za pomiar pojemności
#define MEAS_ENABLE_ISRS TIMSK1 = (1<<ICIE1)|(1<<TOIE1)                //Zezwolenie na przerwania od przechwycenia i przepełnienia licznika
                                                                       //Timer1
#define MEAS_DISABLE_ISRS TIMSK1 = 0                                   //Wyłączenie wszystkich przerwań licznika Timer1
#define MEAS_CLEAR_ISRS_FLAGS TIFR1 = (1<<ICF1)|(1<<TOV1)              //Wyczyszczenie flag przerwań od przechwycenia i przepełnienia
                                                                       //licznika Timer1
#define MEAS_CLEAR_COUNTER TCNT1 = 0                                   //Wyzerowanie licznika Timer1
#define MEAS_START_COUNTER TCCR1B = (1<<ICES1)|(1<<CS10)               //Uruchomienie i konfiguracja licznika Timer1: Preskaler = 1,
                                                                       //przechwycenie na zboczu rosnącym
#define MEAS_STOP_COUNTER TCCR1B = (1<<ICES1)                          //Zatrzymanie licznika Timer1

//Ustawienia komparatora analogowego ACO
#define ANALOG_COM_INPUT_CAP_ENABLE ACSR |= (1<<ACIC)

//Flagi procesu pomiarowego
#define FLAG_IDLE 0                                                    //Flaga bezczynności procesu pomiarowego
#define FLAG_RDY_LOW 1                                                 //Flaga gotowości danych pomiarowych dla mniejszego kondensatora
#define FLAG_RDY_HIGH 2                                                //Flaga gotowości danych pomiarowych dla mniejszego kondensatora
#define FLAG_IN_PROGRESS 3                                             //Flaga trwającego procesu pomiarowego
#define FLAG_OVF 4                                                     //Flaga przekroczenia zakresu pomiarowego
#define FLAG_ADD_DELAY 5                                               //Flaga konieczności uwzględnienia dodatkowego czasu na rozładowanie
                                                                       //kondensatora (dla dużych jego wartości)

//Maksymalna liczba przepełnień licznika Timer1 dla obu wartości rezystancji ładowania kondensatora mierzonego
#define LOW_CAP_MAX_OVFS 27                                            //100 nF
#define HIGH_CAP_MAX_OVFS 57                                           //1000 uF

//Definicje stałych kondensatora zerowego 25pF (dopuszczalnej pojemności pasożytniczej)
#define ZERO_CAP_PF 25                                                 //Jednostka pF
#define ZERO_CAP_PULSES ((ZERO_CAP_PF*1339UL)/100)                     //Inaczej: CpF/0.074653

//Definicje stałych kondensatora referencyjnego 47nF
#define REF_CAP_NF 47                                                  //Jednostka nF
#define REF_CAP_PULSES (REF_CAP_NF*13395UL)                            //Inaczej: CnF/0.000074653

//Definicja stałej kondensatora pierwszego zakresu pomiarowego 99.9nF (liczba impulsów do naładowania do 0.5 VCC)
#define FIRST_STAGE_CAP 999                                            //Jednostka 0.1 nF
#define FIRST_STAGE_CAP_PULSES (1830UL*FIRST_STAGE_CAP)

//Definicje stałych maksymalnego kondensatora, dla którego nie trzeba uwzględniać dodatkowego czasu na jego rozładowanie
#define HIGH_CAP_UF 440                                                //Jednostka uF
#define HIGH_CAP_PULSES (HIGH_CAP_UF*2757UL)                           //Inaczej: CuF/0.00036229

//Flagi funkcji startującej pomiar (wybór zakresu pomiarowego)
#define LOW_CAP 0
#define HIGH_CAP 1


//Prototypy funkcji
void initMeasurement(void);
void startMeasurement(uint8_t capType);
void stopMeasurement(void);


//Zmienna modułu
extern volatile uint8_t measFlag;                                      //Flaga procesu pomiarowego
extern volatile uint8_t Overflows;                                     //Liczba przepełnień licznika Timer1
extern volatile uint32_t Counts;                                       //Liczba zliczonych impulsów
```

Listing 6. Plik nagłówkowy mechanizmu pomiarowego

moment atomowej aktualizacji zmiennych volatile procedury obsługi przerwania mechanizmu multipleksowania. Potrzeba wprowadzenia takiej zmiennej wynikała z konieczności synchronizacji aktualizacji zmiennych (dokonywanej w aplikacji głównej) z pracą funkcji multipleksującej wyświetlacz LED, tak by nie występowało zjawisko „mieszania" zawartości zmiennych w kolejnych „przebiegach" funkcji multipleksującej. Aktualizacja, o której mowa powyżej, następuje po pełnym cyklu multipleksu dla całego wyświetlacza LED. To wszystko, jeśli chodzi o obsługę wyświetlacza – przejdźmy zatem do grupy funkcji odpowiedzialnych za mechanizm pomiarowy. Tradycyjnie, na początek plik nagłówkowy, dzięki któremu porządkujemy późniejszy kod źródłowy czyniąc go bardzo czytelnym, a jednocześnie upraszczając proces ewentualnego wprowadzania zmian. Plik ten, pokazany na **listingu 6**, definiuje główne ustawienia sprzętowe, a także wprowadza niezbędne zmienne. Dalej, na **listingu 7**, znalazło się ciało funkcji odpowiedzialnej za inicjalizację mechanizmu pomiarowego. Kolejne funkcje, których ciała widoczne są na **listingach 8** i **9**, uruchamiają i zatrzymują ów mechanizm.

I na sam koniec dwie funkcje obsługi przerwań: od przechwycenia zawartości licznika (Timer1) oraz od przepełnienia licznika (Timer1), które realizują właściwy mechanizm pomiarowy. Ciała tychże funkcji pokazano na **listingach 10** i **11**.

Powyższa informacja zamyka kwestie implementacyjne. Przejdźmy zatem do schematu montażowego naszego urządzenia, widocznego na **rysunkach 5a** i **5b**.

```
void initMeasurement(void)
{
    //Wybór dolnego zakresu pomiarowego: 100 nF
    //(de facto wielkości rezystora szeregowego)
    MEAS_SELECT_LOW_CAP;
    //Port wyboru zakresu pomiarowego, jako wyjściowy
    MEAS_SELECT_CAP_AS_OUTPUT;
    //Wybór napięcia odniesienia dla komparatora analogowego: 0.17 Vcc
    MEAS_SELECT_LOW_REF;
    //Rozładowanie kondensatora
    MEAS_DISCHARGE_CAP;
    //Podłączenie wyjścia komparatora analogowego do wejścia ICP (Capture) Timera1
    ANALOG_COM_INPUT_CAP_ENABLE;
    //Domyślna wartość flagi pomiarowej
    measFlag = FLAG_IDLE;
}
```

Listing 7. Funkcja odpowiedzialna za inicjalizację mechanizmu pomiarowego

```
inline void startMeasurement(uint8_t capType)
{
    //Ustawienie domyślnych wartości zmiennych procesu pomiarowego
    Overflows = 0;
    measFlag = FLAG_IN_PROGRESS;
    //Wybór zakresu pomiarowego: 100 nF lub 1000 uF
    //(de facto wielkości rezystora szeregowego)
    if(capType == LOW_CAP) MEAS_SELECT_LOW_CAP; else MEAS_SELECT_HIGH_CAP;
    //Wybór napięcia odniesienia dla komparatora analogowego: 0.17 Vcc
    //(plus krótki delay na ustabilizowanie się napięcia)
    MEAS_SELECT_LOW_REF;
    _delay_us(5);
    //Wyzerowanie licznika Timer1
    MEAS_CLEAR_COUNTER;
    //Skasowanie flag przerwań licznika Timer1
    MEAS_CLEAR_ISRS_FLAGS;
    //Uruchomienie przerwań licznika Timer1 (OVF i ICP)
    MEAS_ENABLE_ISRS;
    //Start licznika Timer1 (@8 MHz)
    MEAS_START_COUNTER;
    //Rozpoczęcie ładowania kondensatora
    MEAS_CHARGE_CAP;
}
```

Listing 8. Funkcja odpowiedzialna za uruchomienie mechanizmu pomiarowego

```
void stopMeasurement(void)
{
    //Zatrzymanie licznika Timer1
    MEAS_STOP_COUNTER;
    //Wyłączenie przerwań licznika Timer1 (OVF i ICP)
    MEAS_DISABLE_ISRS;
    //Rozpoczęcie rozładowania kondensatora
    MEAS_DISCHARGE_CAP;
}
```

Listing 9. Funkcja odpowiedzialna za zatrzymanie mechanizmu pomiarowego

```
//Funkcja odpowiedzialna za przechwycenie zawartości licznika Timer1 po naładowaniu kondensatora mierzonego do wartości 0.17 lub 0.5 VCC

ISR(TIM1_CAPT_vect)
{
    static uint32_t T1, T2;  //Liczba zliczonych impulsów dla obu progów napięcia odniesienia: 0.17 i 0.5 Vcc

    //Nastąpiło przechwycenie zawartości licznika Timer1, więc sprawdzamy dla jakiego poziomu napięcia: 0.17 lub 0.5 Vcc
    if(MEAS_LOW_REF_IS_SELECTED)
    {
        //Przechwycenie nastąpiło dla niższego napięcia Vref (0.17 Vcc) więc zapisujemy liczbę zliczonych impulsów
        T1 = ICR1 + (Overflows * 65536);
        //Zmieniamy Vref na 0.5 Vcc (plus krótki delay na ustabilizowanie się napięcia)
        MEAS_SELECT_HIGH_REF;
        _delay_us(5);
        //Skasowanie ewentualnych flag przerwań Timera1
        MEAS_CLEAR_ISRS_FLAGS;
    }
    else
    {
        //Zatrzymujemy pomiar co powoduje zatrzymanie zegara Timer1, wyłączenie przerwań zegara oraz rozładowanie kondensatora
        stopMeasurement();
        //Przechwycenie nastąpiło dla wyższego napięcia Vref (0.5 Vcc) więc zapisujemy liczbę zliczonych impulsów
        T2 = ICR1 + (Overflows * 65536);
        //Obliczenie liczby zarejestrowanych imlulsów licznika Timer1
        Counts = T2-T1;

        //Sprawdzenie, czy zarejestrowana liczba impulsów przekracza zakres dla 99.9 nF i jeśli tak to startujemy proces pomiarowy
        //od nowa ze mniejszym rezystorem pomiarowym
        if(T2 > FIRST_STAGE_CAP_PULSES)
        {
            //Maksymalny, dodatkowy czas na rozładowanie naładowanego w poprzednim kroku kondensatora (rozładowanie do 0.45 V,
            //czyli poniżej 0.17 VCC)
            _delay_us(15);
            //Wznowienie procesu pomiarowego dla nowego zakresu mierzonych pojemności
            startMeasurement(HIGH_CAP);
        }
        //Ustawiamy odpowiednią flagę w zależności od bieżącego zakresu pomiarowego (wielkości mierzonego kondensatora)
        else measFlag = MEAS_LOW_CAP_IS_SELECTED? FLAG_RDY_LOW : FLAG_RDY_HIGH;
    }
}
```

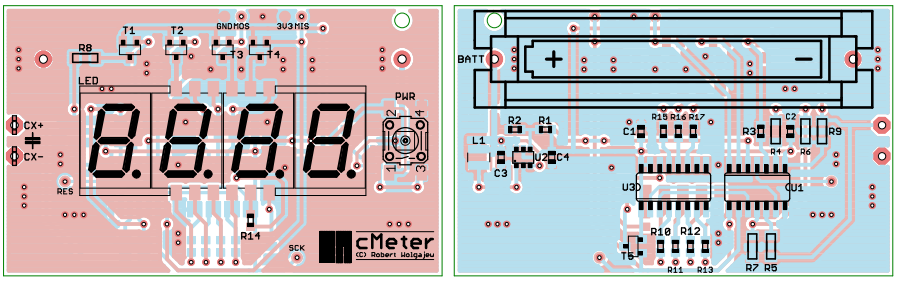
Listing 10. Funkcja obsługi przerwania od przechwycenia zawartości licznika Timer1 realizująca mechanizm pomiarowy

```
//Funkcja odpowiedzialna za policzenie liczby przepełnień licznika Timer1 podczas pomiaru pojemności kondensatora mierzonego

ISR(TIM1_OVF_vect)
{
    ++Overflows;

    if(MEAS_LOW_CAP_IS_SELECTED)
    {
        //Przekroczono dolny zakres pomiarowy (99.9 nF), więc wybieramy mniejszy rezystor ładowania kondensatora
        //i rozpoczynamy proces od nowa
        if(Overflows > LOW_CAP_MAX_OVFS)
        {
            //Zatrzymanie procesu pomiarowego (w tym rozpoczęcie rozładowania kondensatora)
            stopMeasurement();
            //Maksymalny, dodatkowy czas na rozładowanie naładowanego w poprzednim kroku kondensatora (rozładowanie do 0.45 V,
            //czyli poniżej 0.17 VCC)
            _delay_us(15);
            //Wznowienie procesu pomiarowego dla nowego zakresu mierzonych pojemności
            startMeasurement(HIGH_CAP);
        }
    }
    else
    {
        //Przekroczono zakres pomiarowy urządzenia, więc wyświetlamy stosowny komunikat ustawiając bity zmiennej measFlag
        if(Overflows > HIGH_CAP_MAX_OVFS)
        {
            //Zatrzymanie procesu pomiarowego (w tym rozpoczęcie rozładowania kondensatora)
            stopMeasurement();
            //Ustawienie flagi Overflow
            measFlag = FLAG_OVF;
        }
    }
}
```

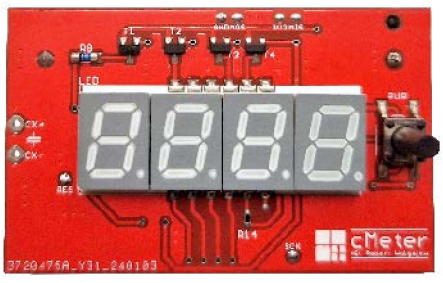
Listing 11. Funkcja obsługi przerwania od przepełnienia licznika Timer1 realizująca mechanizm pomiarowy



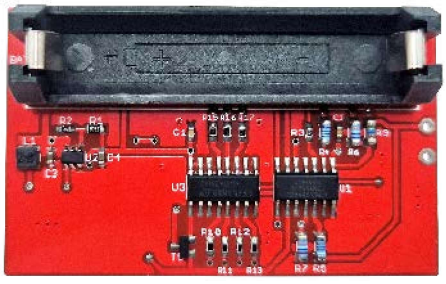
Rysunek 5. Schemat montażowy urządzenia cMeter (a – strona TOP, b – strona BOTTOM)

Jak widać, zaprojektowano bardzo zgrabną, dwustronną, niewielką płytkę drukowaną ze zdecydowaną przewagą elementów SMD lutowanych po obu stronach laminatu. Montaż urządzenia rozpoczynamy od warstwy TOP, na której w pierwszej kolejności przylutowujemy wszystkie półprzewodniki, w tym wyświetlacz LED. 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 z użyciem typowej stacji lutowniczej. Najprostszym sposobem montażu elementów o tak dużym zagęszczeniu wyprowadzeń, niewymagającym jednocześnie posiadania specjalistycznego sprzętu, jest zastosowanie zwykłej stacji lutowniczej, dobrej jakości cyny z odpowiednią ilością topnika oraz dość cienkiej plecionki rozlutowniczej, która umożliwi usunięcie nadmiaru cyny spomiędzy wyprowadzeń układów. Należy przy tym uważać, by nie uszkodzić termicznie tego rodzaju elementów. Następnie lutujemy elementy bierne, po czym przechodzimy na warstwę BOTTOM. Tutaj, podobnie jak poprzednio, montaż rozpoczynamy od przylutowania wszystkich półprzewodników (w tym układów scalonych), po czym montujemy pozostałe elementy bierne oraz gniazdo baterii AAA. W tym momencie wracamy na warstwę TOP, gdzie przylutowujemy przycisk PWR. Na tym etapie urządzenie gotowe jest do uruchomienia. Na **fotografii 1** pokazano zmontowane urządzenie cMeter od strony warstwy TOP, zaś na **fotografii 2** – to samo urządzenie od strony warstwy BOTTOM.

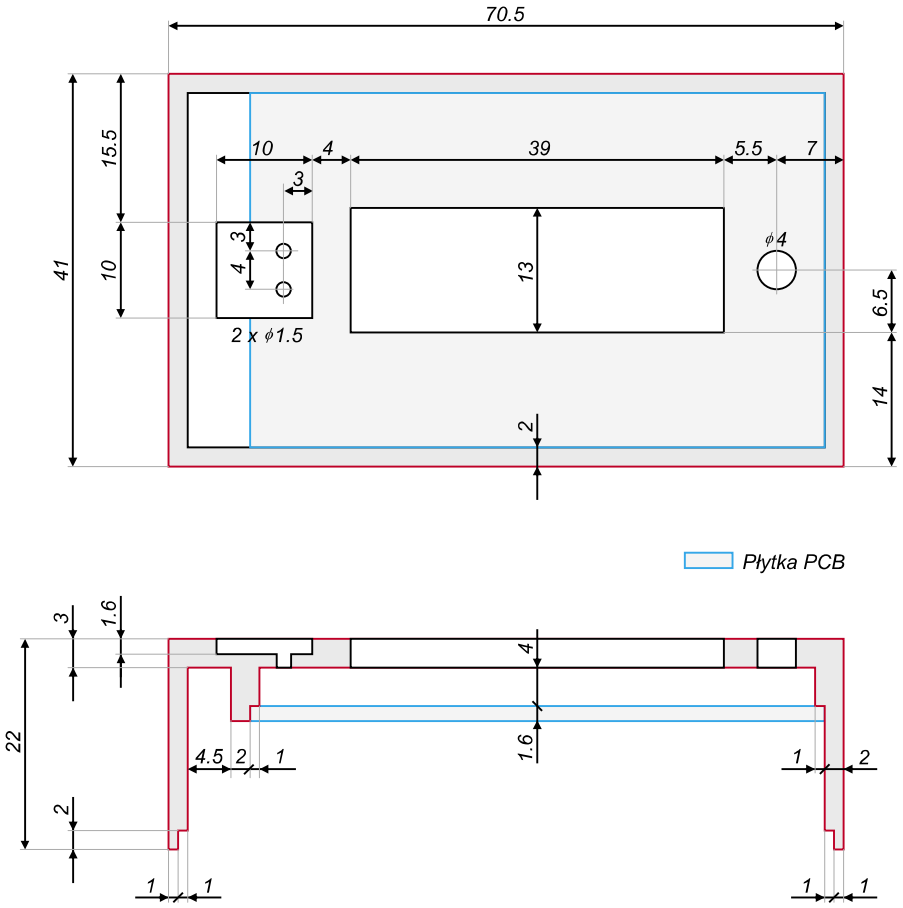
Przejdźmy zatem do opisu obsługi naszego urządzenia. Tu sprawa jest niezmiernie prosta: podczas pierwszego uruchomienia urządzenia, czyli po włożeniu baterii zasilającej do koszyczka, następuje włączenie urządzenia oraz kalibracja wartości zerowej pojemności, co sprowadza się w tym przypadku do pomiaru pojemności pasożytniczej, powinno zatem zostać wykonane przy niepodłączonych do jakiegokolwiek kondensatora stykach pomiarowych. Uzyskana wartość zapisywana jest w pamięci RAM i używana podczas późniejszego procesu pomiarowego. W tym momencie miernik gotowy jest do pracy. Pomiary dokonywane są maksymalnie 2 razy na sekundę, zaś wartość mierzona wyświetlana jest łącznie z jej jednostką (cyfra pierwsza od prawej) na wyświetlaczu LED, przy czym zakres pomiarowy wraz z symbolem wyświetlanej jednostki (p, n, u) jest wybierany automatycznie przez urządzenie. W przypadku kondensatorów o pojemności powyżej 999 µF wyświetlony zostanie napis **high**. Co ważne, przewidziano również możliwość kalibracji naszego miernika, by wyświetlane wartości jak najbardziej odzwierciedlały rzeczywistą wartość mierzonej



Fotografia 1. Zmontowane urządzenie cMeter od strony warstwy TOP



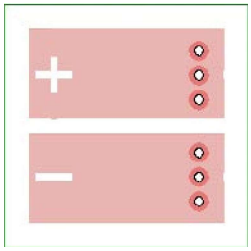
Fotografia 2. Zmontowane urządzenie cMeter od strony warstwy BOTTOM



Rysunek 6. Widok 2D obudowy urządzenia cMeter z zaznaczeniem kluczowych wymiarów

pojemności. W tym celu do styków pomiarowych urządzenia należy dołączyć kondensator wzorcowy o pojemności 47 nF, po czym nacisnąć i przytrzymać przycisk PWR. Miernik wyświetli wówczas napis **CAL** i dokona pomiaru wartości kondensatora wzorcowego – zakładając, że ma on pojemność o dokładnej wartości 47 nF, po czym obliczy stosowny współczynnik korekcyjny i zapisze go w pamięci RAM oraz EEPROM. Dokona tego jednak tylko wtedy, gdy mierzona wartość pojemności znajdzie się w zakresie ±10% wartości 47 nF. Obliczony współczynnik korekcyjny brany będzie następnie pod uwagę przy obliczaniu wartości pojemności mierzonych kondensatorów. Zakończenie pomiaru kondensatora wzorcowego zostanie następnie potwierdzone wyświetleniem napisu **donE**. Z kolei w celu wyłączania i ponownego włączenia urządzenia należy nacisnąć krótko przycisk **PWR** – wyświetli się wówczas napis powitalny **robi**. Warto również podkreślić, że cMeter wyposażono w mechanizm automatycznego wyłączenia (ze względu na potrzebę oszczędzania energii), który aktywuje się po upływie 30 sekund od włączenia urządzenia. Dla porządku dodam, iż urządzenie cały czas monitoruje wartość napięcia baterii zasilającej i przy jego nieodpowiedniej wielkości wyświetla napis **bAtt**, po czym przechodzi do stanu uśpienia.

Omówiliśmy zagadnienia związane z obsługą, przejdźmy zatem do nie mniej ciekawych kwestii dotyczących obudowy urządzenia. Przystępując do prac projektowych, przygotowałem stosowny projekt 2D, którego rysunek z zaznaczeniem wszystkich niezbędnych wymiarów pokazano na **rysunku 6**.



Rysunek 7. Wygląd obwodu drukowanego (od strony warstwy TOP) płytki styków pomiarowych urządzenia cMeter

Rysunek 8. Widok 3D obudowy urządzenia cMeter

Jak widać, w płycie czołowej obudowy (jej ściance górnej) o grubości 3 mm przewidziano „specjalne" wyżłobienie, w które należy wkleić niewielką płytkę drukowaną o wymiarach 10×10 mm, integrującą w sobie styki pomiarowe (pola miedzi). Do płytki tej, od spodu, należy przylutować przewody, które przeciągnąć trzeba do środka obudowy i podłączyć do styków Cx± na płytce drukowanej urządzenia cMeter, zachowując odpowiednią polaryzację. Na **rysunku 7** pokazano wygląd obwodu drukowanego (od strony warstwy TOP) płytki styków pomiarowych, o której mowa powyżej.

Przeanalizujmy zatem konkrety dotyczące projektu 3D obudowy. W tym zakresie, jak to ostatnio u mnie bywa, poprosiłem mojego kolegę **Bartłomieja Wawrzyszko**, zajmującego się hobbystycznie projektami tego rodzaju, o przygotowanie stosownej obudowy, zgodnej z moim projektem 2D, w środowisku pozwalającym na wydruk na drukarce 3D. W efekcie powstał projekt obudowy, którego widok 3D pokazano na **rysunku 8**.

Wspomniana obudowa składa się tak naprawdę z dwóch elementów:

- części górnej (pokazanej na rysunku 8), w której umieszczono otwory na elementy interfejsu użytkownika (okienko wyświetlacza LED i otwór na przycisk PWR) oraz wyżłobienie na płytkę styków pomiarowych
- części dolnej pełniącej funkcję klapki, za pomocą której zamykamy obudowę od dołu (unieruchamiając ją dodatkowo stosownym wkrętem blokującym).

W części górnej naszej obudowy, we wspomnianym powyżej okienku na wyświetlacz LED, należy umieścić przydymioną pleksi o wymiarach 13×39 mm i grubości 3 mm, stanowiącą ochronę wyświetlacza i zapewniającą atrakcyjność wizualną. Dociekliwym Czytelnikom podpowiem, że kupiłem taki kawałek pleksi na znanym portalu aukcyjnym (łącznie z usługą jej wycięcia) za przysłowiowe 3 zł. Wracając do tematu obudowy warto podkreślić, że stosowne pliki z jej projektem (do wydrukowania na drukarce 3D) zostaną zamieszczone na serwerze http://ep.com.pl. Dodatkowo podpowiem, że jeśli nie dysponujecie odpowiednim urządzeniem umożliwiającym wydrukowanie obudowy według załączonych plików, to z powodzeniem możecie to zlecić (i to na naprawdę atrakcyjnych warunkach) jednej z chińskich firm, która znana jest z produkcji obwodów drukowanych w bardzo przystępnych cenach. Na **fotografii 3** pokazano wygląd wspomnianej obudowy wykonanej na drukarce 3D pracującej w technologii FDM (*Fused Deposition Modeling*), w której półpłynne tworzywo sztuczne spaja się pod wpływem wysokiej temperatury i szybko zastyga, tworząc (niemalże) jednolitą strukturę. Niemalże,

Fotografia 3. Widok obudowy urządzenia cMeter wydrukowanej w technologii FDM

Fotografia 4. Widok dolnej części obudowy (klapki) urządzenia cMeter wydrukowanej w technologii SLA

gdyż (co widać na fotografii 3) obudowa wydrukowana w ten sposób ma pewną, nie zawsze akceptowalną, strukturę (poniekąd zależną od jakości samej drukarki).

W celu zobrazowania kontrastu, na zdjęciu tytułowym pokazano wygląd obudowy urządzenia cMeter wydrukowanej na drukarce 3D pracującej w technologii MJF (*Multi Jet Fusion*), polegającej na druku 3D ze sproszkowanych tworzyw sztucznych (poliamidów), poprzez selektywne natryskiwanie na nie lepiszcza (sklejającego ze sobą poszczególne warstwy modelu) i zgrzewania ich w wysokiej temperaturze, co skutkuje ich trwałym zespojeniem. Atutem wydruków 3D z zastosowaniem technologii MJF jest wysoka wytrzymałość mechaniczna produkowanych części. Uzyskiwana jest ona dzięki jednolitej strukturze, która ma 100-procentowe wypełnienie. Takie drukowanie 3D sprawia, że można uzyskać dowolne części, nawet o wysokim stopniu skomplikowania elementów, a koszt wydruku uzależniony jest wyłącznie od ilości zużytego materiału. Należy przy tym podkreślić, że powstające produkty mają wysoką powtarzalność, z dokładnością wymiarową do 0,2 mm. Metoda MJF w drukowaniu 3D pozwala na wytwarzanie funkcjonalnych części, o gładkiej powierzchni, niskiej porowatości i dowolnej geometrii. Wspomniana powyżej obudowa została wydrukowana dzięki usłudze jednej z dalekowschodnich firm produkujących obwody drukowane, o czym wspomniałem już wcześniej. Szczerze polecam tego typu rozwiązanie. Na **fotografii 4** pokazano wygląd dolnej części obudowy (klapki) wydrukowanej w SLA, czyli technologii, która bazuje na procesie fotopolimeryzacji (podobnie jak technologia DLP – ang. *Direct Light Processing* – z którą ma wiele wspólnych cech). Budowany obiekt powstaje w tym przypadku wskutek selektywnego utwardzania żywicy fotopolimerowej światłem lasera. Metoda ta (nazywana pierwotnie stereolitografią) jest najstarszą metodą druku 3D, uznawaną przez wielu za jedną z najbardziej ekonomicznych metod przyrostowych. Drukowane elementy łączą w sobie wysoką dokładność z dużą gładkością powierzchni. Niestety są mało odporne na wysokie temperatury i dość często już przy temperaturach rzędu 50°C stają się ciągliwe.

Robert Wołgajew, EP