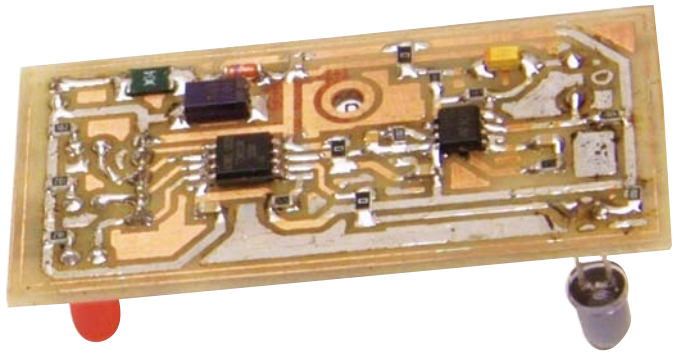


„Dotykowa” błyskotka

Do czego to służy?

Tytułowe urządzenie jest prostą zabawką, która w założeniu ma służyć jako ozdoba. Układ został wyposażony w diody LED, których jasność świecenia jest proporcjonalna do odległości ręki od czujnika podczerwieni. Kiedy odległość ta jest zbyt duża, to jasność diody jest określana przez ciąg pseudolosowy.

Urządzenie może stanowić podstawę do budowy różnorodnych „magicznych” przedmiotów, takich jak kolorowe szklane kule, które zmieniają swoją jasność w zależności od odległości dłoni. Nic nie stoi na przeszkodzie, aby zbudować dwa lub trzy tego typu układy i w każdym z nich zastosować inne kolory diod LED. Pozwoli to uzyskać wielobarwne efekty zmieniające się losowo lub proporcjonalnie do odległości ręki.

Do poprawnej pracy niezbędny jest zaprogramowany mikrokontroler. Plik wynikowy dostępny jest na stronie Elportalu. Osoby mniej doświadczone mogą poprosić o pomoc w zaprogramowaniu kolegę lub zakupić układ z wgranym softem w sklepie AVT. Bardziej zaawansowani znajdą na stronie kod źródłowy, który może zostać zmodyfikowany w sposób uwzględniający indywidualne potrzeby. Dodatkowo, obecny jest jeszcze jeden plik wynikowy ze słowami *no random*, który nie generuje efektów losowych – diody po odsunięciu ręki zostaną wygaszone.

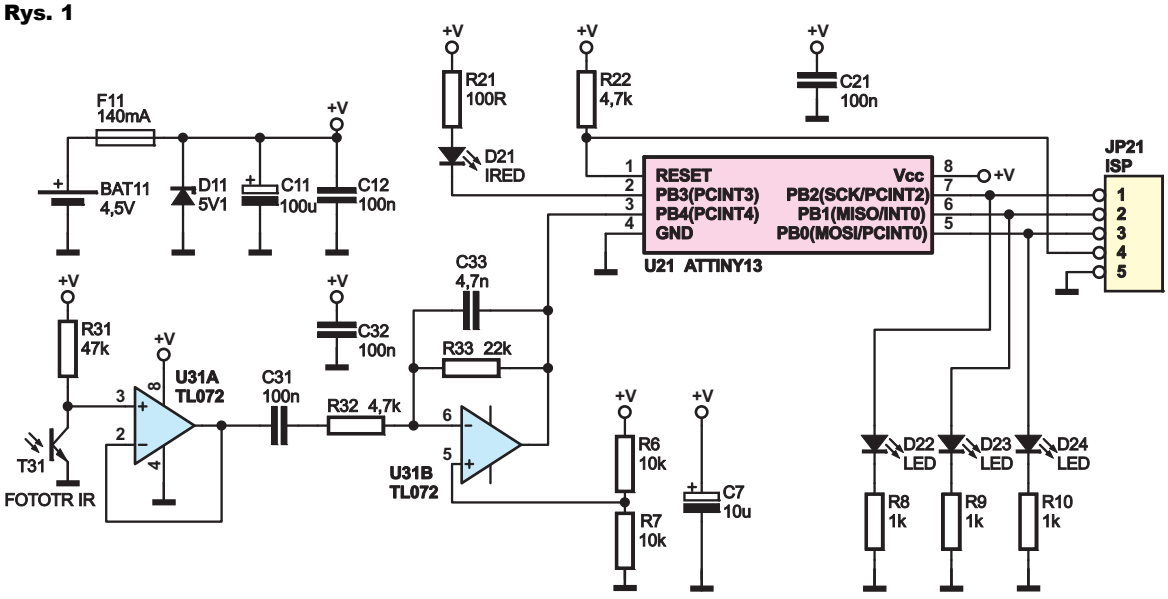
Jak to działa?

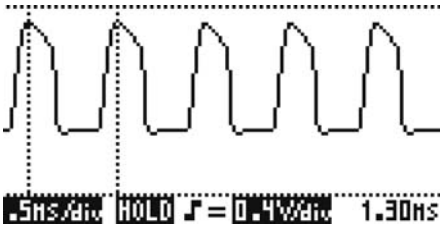
Schemat urządzenia pokazano na **rysunku 1**. Jako źródło zasilania przewidziane zostały trzy popularne „paluszki”. Możliwe jest także zastosowanie stabilizowanego zasilacza o zakresie napięć 4...5V. Blok zasilający zawiera bezpiecznik polimerowy, który ma ograniczyć pobór prądu na wypadek zwarcia. Dioda Zenera zabezpiecza przed zbyt dużym napięciem na wejściu oraz odwrotną polaryzacją, w przypadku której spadek napięcia na diodzie wyniesie 0,7V. W efekcie do mikrokontrolera oraz wzmacniacza zostanie doprowadzone napięcie –0,7V, które nie powinno uszkodzić tych elementów. Kondensatory C11 oraz C12 pełnią funkcję filtrującą.

Cała inteligencja urządzenia skupiona została w mikrokontrolerze z rodziny AVR – ATtiny13. Posiada on tylko osiem wyprowadzeń, co pozwala zmniejszyć wymiary płytki drukowanej. Dodatkowo pamięć RAM umożliwia przygotowanie oprogramowania w języku C++. Do mikrokontrolera dołączona jest bezpośrednio dioda IR, która jest włączana poprzez wystawienie stanu niskiego na port PB3. Rezystor R21 ogranicza płynący przez nią prąd, nie dopuszczając do uszkodzenia. Kondensator C21 tłumi ewentualne zakłócenia w dodatnim napięciu zasilającym. Diody D22...D24, jak łatwo się domyślić, są tymi diodami, które zmieniają swoją jasność proporcjonalnie do odległości dłoni od urządzenia. Po dołączeniu diody IR i odbiornika IR pozostały trzy wolne porty i postanowiłem je zagospodarować – stąd trzy diody LED. Ma to swoje zalety, bo dzięki temu całość świeci mocniej bez konieczności dolutowywania dodatkowych diod w pająku.

Ostatni blok widoczny na schemacie składa się z elementów, które mają sygnatury zaczynające się literą i cyfrą 3 (np. U31, C31, R32, etc.). Tworzą one prosty filtr pasmowy na zakres częstotliwości od 0,34kHz do 1,54kHz. Dodatkowo sygnał wzmacniany jest o 6,7dB (około 4,7 razy). W pierwszym etapie wzmocnienie było znacznie większe, a pasmo znacznie węższe, co powodowało praktycznie dwustanową pracę toru przetwarzania podczerwieni – sygnał był nieobecny albo

Rys. 1





Rys. 2

miał maksymalną amplitudę. Po zmniejszeniu wzmocnienia całość zaczęła pracować znacznie lepiej. Jakość wzmacniacza jest dyskusyjna, gdyż przebieg oglądany na oscyloskopie (**rysunek 2**) odbiega od sygnału prostokątnego (takim kluczowana jest dioda IR), ale nie ma to większego znaczenia, gdyż liczy się wartość szczytowa na wyjściu wzmacniacza. Z rysunku 2 można odczytać, że okres sygnału wynosi 1,3ms. Można to przeliczyć na częstotliwość według prostego wzoru: f = 1/T. Otrzymamy wtedy około 800Hz, które mieści się w paśmie przenoszenie wzmacniacza.

Oprogramowanie

Podstawę działania urządzenia stanowi oprogramowanie napisane w języku C++ w środowisku WinAVR. Zawartość głównego plik (*main.cpp*) przedstawiono na **listingu 1**. Pierwszym krokiem jest zaznaczenie, jaką częstotliwość ma sygnał zegarowy za pomocą parametru *F_CPU*. Jest to wymagane do prawidłowego odliczania czasu poleceniem *_delay_ms()*.

Następnie dołączane są standardowe biblioteki i dwa pliki nagłówkowe przygotowane specjalnie do tego urządzenia. Pierwszy z nich to *device.h* definiujący klasę o tej samej nazwie. Jej zadaniem jest skonfigurowanie portów i dostarczenie funkcji do sterowania diodami LED diodą nadawczą podczerwieni oraz mierzącej napięcie na wyjściu wzmacniacza operacyjnego.

Drugi plik nagłówkowy odpowiada za zdefiniowanie klasy *generator*. W jej wnętrzu zawarta jest funkcja, która generuje przebieg około 800Hz przeznaczony do kluczowania diody IR oraz kod odpowiedzialny za programową realizację układu PWM sterującego diodami LED. Oba generatory wykorzystują przerwanie pochodzące od licznika T0. Obsługa przerwania nie jest trudna i wymaga jedynie skonfigurowania rejestrów licznika, włączenia systemu przerwań od T0, przerwania globalnego i dodania funkcji *SIGNAL* ze stałą w ramach argumentu określającą rodzaj przyjmowanego przerwania. Jak widać, wewnątrz tej funkcji znajduje się wywołanie kodu pochodzącego z klasy *generator*.

Pętla główna nie jest przesadnie skomplikowana. Pierwszy etap polega na określeniu, w jakiej odległości znajduje się ręka. Można to stwierdzić na podstawie wartości szczytowej sygnału na wyjściu wzmacniacza. Przebieg obserwowany na wejściu przetwornika ADC nie jest stały, wartość szczytowa pojawia się tylko na moment i nie wiadomo kiedy. Jak ją zatem znaleźć? Można zebrać kolekcję próbek i wśród nich szukać największej. Takie rozwiązanie zastosowano. Zebranie 255 próbek zajmie około 7ms, taką wartość w przybliżeniu wskazał oscyloskop. Oznacza to, że pomiarami obejmujemy pięć pełnych okresów sygnału, wśród których na pewno znajdzie się przynajmniej pięć maksimów. Próbkowanie nie odbywa się z nieskończenie małym odstępem, więc nie zawsze idealnie trafimy „na górkę", ale minimalne odchylenie w prawo bądź w lewo nie robi większej różnicy w tym zastosowaniu. Z całej tej kolekcji wybieramy jeden, największy wynik i zapisujemy do zmiennej *wynik*.

W zależności od odległości ręki od toru podczerwieni wartość zmiennej *wynik* będzie się zmieniać. W jakich granicach? Nie sposób tego przewidzieć, gdyż wpływ ma na to zbyt wiele czynników: charakterystyka diody, charakterystyka fototranzystora, napięcie zasilania, tolerancja elementów, etc. Możliwe jest jednak obejście tego problemu przez dynamiczne wyznaczanie wartości minimalnej i maksymalnej. Po wybraniu z kolekcji próbek jednej, największej sprawdzamy, czy jest ona mniejsza od zmiennej *min* lub większa od zmiennej *max*. Zmienne te przechowują wartości dotychczas uważane za najmniejsze i największe. Po stwierdzeniu, że *wynik* wykracza poza ten zakres, następuje jego zmiana. Odbywa się ona poprzez zapis nowej wartości do zmiennej *min* bądź *max*. Od tej pory jest to nasz nowy zakres.

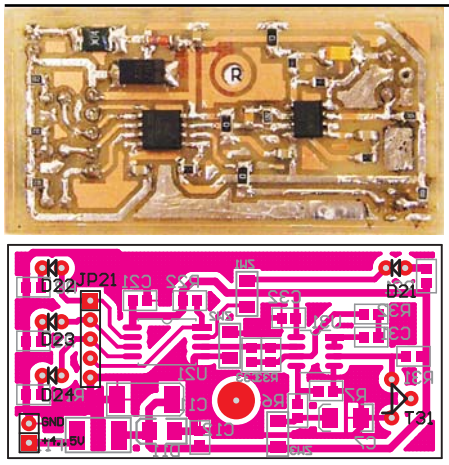
Rzadko się zdarzy, że diody LED zostaną całkowicie wyłączone, ponieważ zmierzona wartość musiałaby odpowiadać wartości zmiennej *min*. Zmienna *min* posiada najmniejszą, możliwą wartość jaka pojawiała się na wyjściu wzmacniacza od momentu uruchomienia urządzenia. Statystycznie zjawisko to nie powtórzy się zbyt często. Rozwiązaniem tego problemu jest wprowadzenie pewnego marginesu, który jest (procentowo) zapisany w zmiennej *margines*. Jego zadanie polega na przesunięciu dolnego progu i utworzeniu „strefy martwej". Dzięki jej obecności urządzenie jest w stanie wyłączyć diody LED. Warto zwrócić uwagę na fakt, że margines jest procentowy, co i tym razem nie wymaga znajomości konkretnych wartości progów.

W zasadzie dyskutuję sobie o granicach, w jakich zmienia się napięcie wejściowe, ale po co? Czy nie można po prostu przepisać zmierzonej wartości od razu do PWM? Niestety nie jest tak dobrze... Napięcie to nie spada do zera, więc mielibyśmy ZAWSZE niezerową wartość i diody świeciłyby cały czas. Druga zła wiadomość jest taka, że zmiany jasności byłyby znacznie mniejsze i mniej zauważalne. Należy w związku z tym przyjąć, że minimalny sygnał występujący na wejściu (po uwzględnieniu wspomnianego marginesu) daje zawsze zerowe wypełnienie (diody wyłączone), a maksymalny sygnał, jaki może pojawić się na wejściu, dostarcza stuprocentowego wypełnienia. Co z wartościami pośrodku? Tutaj pojawia się odrobina matematyki, czyli prosty wzór na skalowanie wartości napięcia na wypełnienie przy uwzględnieniu wyznaczonych uprzednio granic (wartości zmiennych *min* oraz *max*). Załatwiają to następujące instrukcje:

*temp=[wynik–min–(min*margines/100)]* 255;*
temp = temp / (max-min) ;

Listing 1

```
#define F_CPU 4800000UL
//----------------------------------------
#define sbi(port, bit) port|=(1<<bit)
#define cbi(port, bit) port&=~(1<<bit)
#include <avr/io.h> //definicje rejestrow uK
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/crc16.h>
#include <util/delay.h>
#include "src/device.h"
#include "src/generator.h"
//----------------------------------------
SIGNAL(SIG_OVERFLOW0){
  gen.interrupt() ;
}
//----------------------------------------
int main(){
  //zmienne
  unsigned int temp = 0 ;
  unsigned int wynik = 0 ;
  unsigned int losowanie = 0 ;
  unsigned int max=0, min=255 ;
  unsigned int margines = 10 ; //w procentach
  //petla glowna
  while(1){
    //znajdwanie lokalnego maximum - zbieranie probek trwa 7ms
    wynik = 0 ;
    for(unsigned char i=0 ; i<255 ; i++){
      temp = dev.adc() ;
      if(wynik<temp){wynik=temp;}
    }
    //okreslenie granic
    if(max<wynik){max=wynik;}
    if(min>wynik){min=wynik;}
    //wyznaczenie wypełnienia
    if(wynik > (min + (min*margines/100)) ){
      temp = (wynik - min - (min*margines/100) ) * 255 ;
      temp = temp / (max-min) ;
    }else{
      //ustaw nowa wartosc - losowe migotanie
      losowanie = _crc16_update(losowanie, 1);
      temp = losowanie&0x00FF;
        //---------------
      //no random
      //temp = 0 ;
        //---------------
      //opoznienie
      _delay_ms(20) ;
    }
    //sterowanie generatorem PWM
    gen.pwmSet(temp&0x00FF) ;
  }
}
```



Rys. 3

Ich interpretację pod kątem matematycznym pozostawiam dociekliwym Czytelnikom. Wartość 255 w tym wypadku oznacza 100% wypełnienia. Wyrażenie **margines/100* odpowiada pomnożeniu przez ułamek. Co prawda można zastosować liczby zmiennoprzecinkowe, ale: zwiększają się zapotrzebowanie na pamięć programu i RAM oraz potrzeba więcej czasu na obliczenia. Wyrażenie to jest oczywiście prostą sztuczką, która bazuje na fakcie, że kompilator zachowuje kolejność wykonywania działań. Wynika ona z prostej obserwacji: x% to przecież x setnych, co można zapisać ułamkiem x/100. Działania na ułamkach polegają na mnożeniu liczników i mianowników obu liczb, więc mnożymy liczniki (**margines*) i na końcu dzielimy przez mianownik (/100). Tracimy w tym wypadku część wyniku na skutek zaokrąglenia, ale cóż... Nic za darmo:).

Kiedy napięcie wejściowe jest na tyle małe, że mieści się w przyjętym marginesie, zależnie od wersji oprogramowania, diody są wygaszane bądź uruchamiane jest losowe wyznaczanie wypełnienia (efekt przypomina trochę „świeczkę"). Na Elportalu znajdują się dwa pliki wynikowe, plik bez losowej animacji nazywa się *main_no_random.hex*.

Za efekt pseudolosowy odpowiedzialna jest funkcja *_crc16_update()*. Jest ona programową implementacją rejestru LFSR, który normalnie służy do zabezpieczania transmisji przed błędami, ale jest przy okazji generatorem pseudolosowym. Więcej informacji o rejestrach LFSR można znaleźć w czwartej części kursu CPLD. Generowaniu liczb pseudolosowych towarzyszy opóźnienie 20ms, aby zmiany nie były zbyt szybkie, gdyż wtedy przestałyby być zauważalne. Zastosowano tutaj rejestr 16-bitowy, aby wydłużyć okres, po jakim ciąg pseudolosowy ulegnie powtórzeniu.

Ostatnia instrukcja w programie, jak łatwo się domyślić, zapisuje wyznaczoną wartość wypełnienia do klasy realizującej generator PWM.

Montaż i uruchomienie

Układ można zmontować na płytce drukowanej pokazanej na **rysunku 3**. Jak wspomniano na początku, do poprawnej pracy urządzenia wymagane jest wgranie oprogramowania. Można je znaleźć na Elportalu i wykorzystać bez konieczności wprowadzania zmian. Programowanie może się odbywać w systemie za pomocą złącza JP21. Należy mieć na uwadze, że do wykorzystywanych do tego celu portów przyłączone są diody LED. W niektórych przypadkach może to uniemożliwić programowanie, dlatego warto to zrobić przed ich wlutowaniem.

Układ w zasadzie jest przeznaczony do wbudowania w jakiś obiekt, np. szklaną kulę, aby uzyskać w ten sposób ozdobę. Podczerwień może ulegać odbiciu od szkła bądź pleksiglasu, dlatego warto zaopatrzyć się w kawałek czarnej rurki termokurczliwej i nasunąć ją na diodę IR. Nie powinna ona być zbyt długa, ale wskazane byłoby, aby dotykała ścianki obudowy.

Fototranzystor musi być przeznaczony na pasmo podczerwieni, gdyż układ jest wtedy mniej podatny na wpływ zewnętrznego oświetlenia. Praca w warunkach silnego, sztucznego oświetlenia jest niewskazana, gdyż układ nie reaguje wtedy prawidłowo. Jest to spowodowane nieidealną charakterystyki fotorezystora, która „zahacza" o zakres światła widzialnego.

Wykaz elementów

Rezystory

R6,R7	10kΩ 2012 (0805)
R8-R10	1kΩ 2012 (0805)
R21	100Ω 2012 (0805)
R22,R32	4,7kΩ 2012 (0805)
R31	47kΩ 2012 (0805)
R33	22kΩ 2012 (0805)

Kondensatory

C7	10µF obudowa A, SMD
C11	100µF obudowa D, SMD
C12,C21,C31,C32	100nF 2012 (0805)
C33	4,7nF 2012 (0805)

Półprzewodniki

D11	5V1
D21	IRED
D22-D24	LED
T31	Fototranzystor IR
U21	ATtiny13
U31	TL072

Pozostałe

F11	polimerowy140mA 1812 SMD
BAT11	3x1,5V
JP21	5-PIN

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-2913.

Możliwości zmian

Czytelnicy mogą zmienić parametry oprogramowania, takie jak margines „strefy martwej", parametry pomiarów, etc. Zmiany te są dopuszczalne, ale nie są konieczne do prawidłowej pracy. Domyślne oprogramowanie ma zestaw wartości, które zostały uznane przeze mnie za optymalne.

Jakub Borzdyński
jakub.borzdynski@elportal.pl