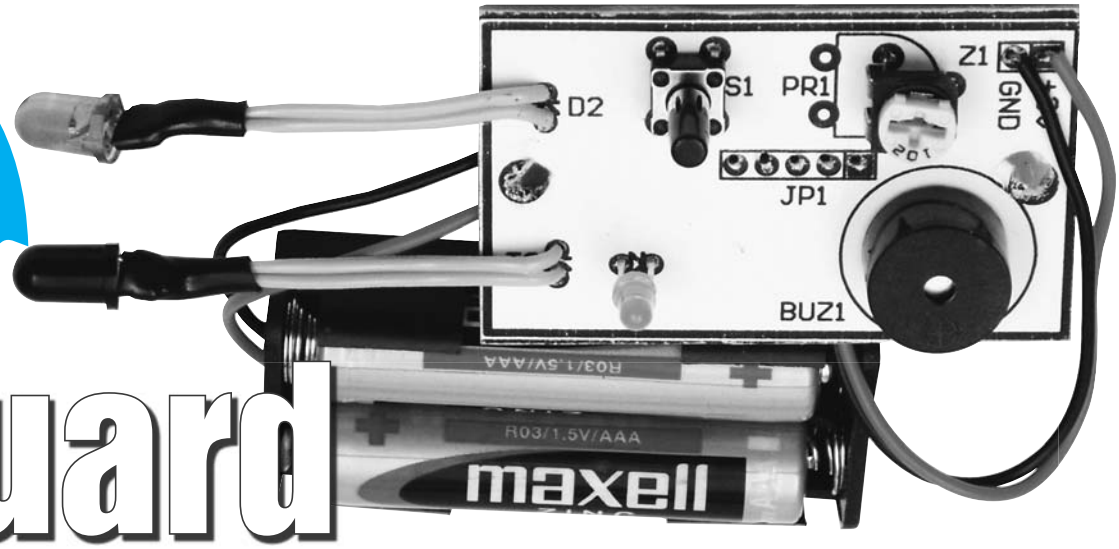
kit 2910 AVT



IRguard – alarm turystyczny

Do czego to służy?

Urządzenie nazwane przeze mnie IRguard jest prostym systemem ochrony, który może znaleźć zastosowanie w czasie wycieczek. Pomysł jego budowy pojawił się podczas podróży pociągiem. Do celu pozostała mi godzina, chciało mi się spać, ale siedziałem sam w przedziale. Pomyślałem wtedy, że bezpieczniej byłoby zasnąć, gdybym mógł się obudzić po wejściu innego pasażera lub konduktora do przedziału. Powstała wtedy koncepcja zaprojektowania układu pełniącego funkcję radaru. Wysłana wiązka podczerwieni ulegałaby odbiciu od drzwi bądź klamki i wracała do odbiornika. Po otwarciu drzwi sygnał nie uległby odbiciu i urządzenie włączyłoby alarm. Mając w głowie taką właśnie koncepcję, przystąpiłem do projektowania.

Układ jest bardzo prosty, więc powinien również zainteresować najmłodszych Czytelników. Na stronie Elportalu można znaleźć gotowy plik wynikowy potrzebny do zaprogramowania mikrokontrolera. W razie problemów warto poprosić o radę bardziej doświadczonego kolegę lub kupić zaprogramowany układ (lub cały kit) w sklepie AVT. Dla bardziej doświadczonych osób udostępniłem także pełny kod źródłowy.

IRguard być może znajdzie inne zastosowania. Może on bowiem pilnować w nocy bagażu (zabranie torby sprawi, że wiązka nie ulegnie odbiciu i włączy się alarm) oraz chronić np. namiot monitorując wejście. Czytelnicy zaproponują zapewne jeszcze inne, ciekawe pomysły.

Jak to działa?

Schemat urządzenia przedstawiono na **rysunku 1**. Najważniejszym elementem jest mikrokontroler ATtiny44. Został on wybrany głównie ze względu na nietypową liczbę wyprowadzeń. Okazuje się, że 5 portów (obudowy so8) to za mało, a 17 (so20) to za dużo. Optymalnym kompromisem był wybór układu w obudowie z 14 wyprowadzeniami. Nie jest on może obecnie tak popularny jak inni członkowie rodziny AVR, ale jest bez problemu dostępny w polskich sklepach internetowych i niniejszy projekt jest dobrą okazją do jego bliższego poznania. Przy zakupie tego elementu należy KONIECZNIE zwrócić uwagą czy posiada on w nazwie literkę „V", która oznacza możliwość pracy z napięciem 1,8V. W przeciwnym wypadku konieczne będzie zasilenie urządzenia napięciem minimum 2,7V.

Obecna na schemacie dioda LED (D3) jest przeznaczona do wizualizacji stanu pracy urządzenia, które może znajdować się w jednym z trzech trybów (o tym za chwilę). Dioda IRED (D2) służy do emitowania wiązki podczerwieni, a fototranzystor T2 pracujący w zakresie podczerwieni służy do jej odbierania (po odbiciu od monitorowanego obiektu).

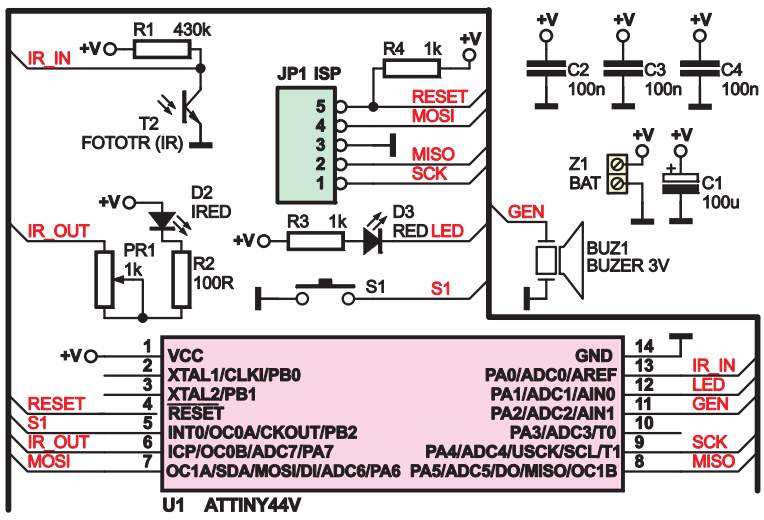
Przycisk S1 przeznaczono do sterowania pracą urządzenia. Jest on podłączony do wejścia przerwania INT0, co umożliwi wprowadzanie AVR-a w stan głębokiego uśpienia i następne jego wybudzanie.

Potencjometrem PR1 można określić natężenie emitowanej podczerwieni poprzez regulację prądu diody IR. W wielu przypadkach nie będzie potrzeby jego montowania i wystarczy zworka. Wynika to z faktu, że zasięg toru podczerwieni wynosi kilka do kilkunastu centymetrów i zmniejszenie prądu diody IR znacznie zmniejszy zasięg (ale jednocześnie pozwoli znacznie wydłużyć czas pracy baterii).

Kondensatory C1...C4 służą do filtracji napięcia zasilania i zmniejszają zakłócenia występujące w ścieżce doprowadzającej napięcie do poszczególnych elementów układu.

Chciałbym przy okazji wspomnieć, że pierwsze wersja urządzenia miała pro-

Rys. 1 Schemat ideowy



stą przetwornicę do sterowania brzęczyka. Okazało się jednak, że buzera nie da się wyłączyć, gdyż potrafi on pracować przy napięciu 1,8V (sic!). Tak więc zmuszony zostałem wykonać kolejną wersję płytki :(. Być może Czytelnicy będą się zastanawiać, czemu nie byłem „sprytny" i nie wstawiłem buzera na 12V. Otóż wstawiłem. Efekt był ten sam – nie można go było wyłączyć. Warto o tym pamiętać, projektując własne urządzenia.

Obsługa urządzenia

Jak wspomniano wcześniej, urządzenie ma trzy tryby pracy:
– wstrzymania (HALT) – dioda LED w ogóle nie świeci,
– oczekiwania (WAIT) – dioda LED szybko pulsuje,
– pracy (WORK) – dioda sporadycznie, krótko błyska.

Tryb wstrzymania jest uruchamiany przez przytrzymanie przycisku S1 przez parę sekund (około 5). Mikrokontroler wchodzi wtedy w tryb oszczędzania energii i nie pobiera prądu. Kiedy urządzenie nie jest używane, powinno pozostawać w tym właśnie trybie.

Funkcja oczekiwania pozwala sprawdzić poprawność pracy. Kiedy monitorowany obiekt jest poza zasięgiem podczerwieni, dioda pulsuje, w przeciwnym wypadku świeci ciągłym światłem. Funkcja ta umożliwia sprawdzenie, czy alarm został poprawnie zainstalowany bez uruchamiania buzera (w nocy nie obudzimy dzięki temu współlokatorów).

Naciśnięcie S1 (krótkie) spowoduje „uzbrojenie" alarmu. Tym razem po wystąpieniu stanu alarmowego brzęczyk zostanie włączony. Kolejne naciśnięcie przycisku (około 1 sekundy) spowoduje przejście do stanu oczekiwania. Z tego miejsca możliwe jest także „uśpienie" procesora poprzez przytrzymanie S1 przez parę sekund.

Na **rysunku 2** pokazano schematyczny opis przejścia do poszczególnych stanów.

Oprogramowanie

Do poprawnej pracy układu niezbędne jest oprogramowanie. Jego pełną wersję można znaleźć w Elportalu. W tym miejscu chciałbym przybliżyć, jak ono funkcjonuje.

Program został napisany w języku C++ za pomocą darmowego środowiska WinAVR. Dla ułatwienia sobie pracy i zwiększenia przejrzystości utworzone zostały dwie klasy.

Pierwsza z nich, *device*, dostarcza funkcji obsługujących elementy takie jak: dioda LED, dioda IR, buzer oraz przycisk. Zawierają one bardzo prosty kod, z wyjątkiem ostatniej. Konieczność określenia, czy S1 został wciśnięty na długo, czy krótko, wymusiła rozbudowanie tej procedury. Jej kod pokazano na **listingu 1**. Na początku tworzona jest zmienna *czas*, która zlicza interwały czasowe. Znajdująca się dalej pętla *while* nie zostanie wykonana, jeżeli przycisk jest puszczony i dzięki temu zmienna *czas* będzie miała wartość zero. Stanowi to informację, że przycisk nie został w ogóle naciśnięty. Gdyby jednak S1 został wciśnięty, to rozpocznie się zwiększanie wartości zmiennej o jeden co 20ms. Zwłoka czasowa ma jeszcze jedno zadanie – wyeliminować drgania zestyków przycisku. Instrukcja *if* zapobiega przepełnieniu zmiennej, aby uniknąć pomyłki przy określanie jak długo S1 był wciśnięty. Ostatnie trzy instrukcje zwracają odpowiedni kod w zależności od tego, jak długo przycisk był przytrzymywany.

```
//zwraca stan przycisku
char device::button(){
  //zmienna do zliczania czasu
  unsigned char czas = 0 ;

  //okresl czas nacisniecia
  while(bit_is_clear(PINB, 2)){
    _delay_ms(20) ;
    if(czas<250){
      czas++ ;
    }
  }
  //czy przytrzymano dluzej niz dwie sekundy ?
  if(czas==0){return 0;} //nie nacisnieto przycisku
  if(czas>100)  {return DEVICE_S1_LONG_PUSH;}
  //dlugie przytrzymanie
  return DEVICE_S1_SHORT_PUSH ;
}
```

Listing 1

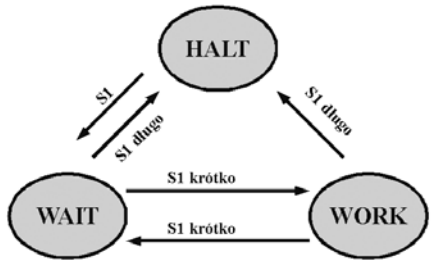
Listing 2

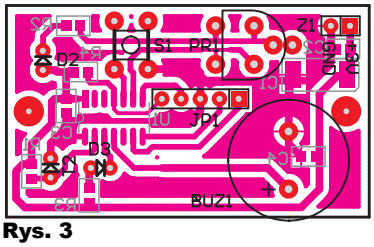
```
while(1){
  char s=dev.button() ;//sprawdz stan przycisku
  //obsluga czujnia podczerwieni
  if(czujnikIR.pomiar()){
    radar = IR_FILTER ; //strefa ochronna
  }else{
    radar -= 1*(radar!=0) ;}
    switch(automat){
    case STATE_HALT:
      //przejsce w stan wstrzymania
      sbi(MCUCR, SM1);//ustaw stan wstrzymania
      cbi(MCUCR, SM0);
      sbi(MCUCR, SE) ;
      //wylacz podzespoly
      dev.led(0) ;
      dev.ired(0) ;
      dev.buzzer(0) ;
      _delay_ms(500) ; //czekaj
      sbi(GIMSK, INT0) ; //odblokuj przerwanie zewnetrzne
      asm("sleep") ; //uspij procesor
      automat = STATE_WAIT  ; //przejdz do stanu ozekiwania
      s = dev.button() ; //czekaj na puszczenie klawisza
      s = 0 ; //skasuj odczytane polecenie
      break ;
    case STATE_WAIT:
      //obsluga klawiatury
      if(s==DEVICE_S1_LONG_PUSH ){automat=STATE_HALT;}
      if(s==DEVICE_S1_SHORT_PUSH){automat=STATE_WORK;}
      //sprawdzanie odbicia - obsluga diody LED
      if(radar!=0){
        dev.led(1) ;
      }else{
        dev.led(delay&0x10) ;}
      break ;
    case STATE_WORK:
      //obsluga klawiatury
      if(s==DEVICE_S1_LONG_PUSH ){automat=STATE_HALT;}
      if(s==DEVICE_S1_SHORT_PUSH){automat=STATE_WAIT;}
      dev.led(delay>250) ; //obsluga diody LED
      //obsluga czujnika podczerwieni
      if(radar == 0){
        dev.buzzer(1) ;
        _delay_ms(500) ;
        dev.buzzer(0) ;
      }
      break ;
  }
  //zwloka czasowa i odliczanie czasu
  _delay_ms(4) ;
  delay++;
}
```

Druga klasa została nazwana *sensor*. Jej rola sprowadza się do określenia, czy wiązka podczerwieni ulega odbiciu, czy też nie. Konstruktor klasy przygotowuje przetwornik do pracy poprzez ustawienie rejestrów. Napięcie odniesienia jest domyślne – masa oraz plus zasilania. Funkcja *adc* rozpoczyna pomiar, oczekuje, aż się on zakończy i zwraca rezultat w postaci zmiennej 16-bitowej.

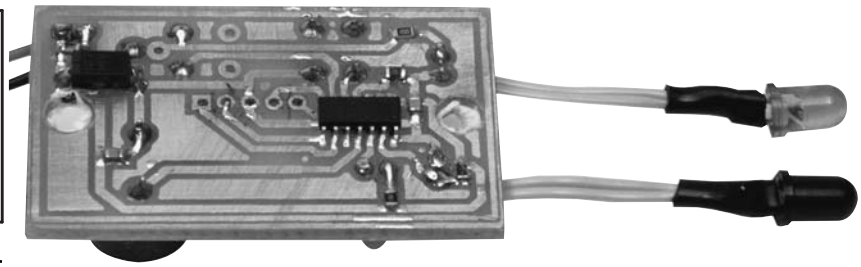
Najważniejsza jednak jest funkcja *pomiar*. Zwraca ona wartość 1-bitową typu *true* lub *false* i informuje, czy wiązka ulega odbiciu, czy też nie. W zasadzie taka informacja jest wystarczająca, gdyż nie musimy wiedzieć, czy odbicie jest słabsze, czy silniejsze, nie ma to dużego znaczenia. Idea pomiaru jest bardzo prosta. Na początku włączana jest dioda IR i mierzone jest napięcie doprowadzone do przetwornika przez sygnał *IR_IN*. Następnie wyłączana jest dioda IR i ponownie mierzone jest napięcie. Gdyby podczerwień ulegała odbiciu, to wynik pierwszego pomiaru będzie mniejszy, gdyż fototranzystor będzie częściowo otwarty. Różnica obu pomiarów, przechowywanych w zmiennych *min* oraz *max*, decyduje o tym, czy mamy do czynienia z odbiciem, czy nie. Jeżeli różnica ta będzie większa od stałej *IR_SENSOR_DELTA*, wtedy mamy do czynienia z odbiciem wiązki podczerwieni. Zmieniając wartość parametru *IR_SENSOR_DELTA*, można zwiększyć czułość odbiornika. Nie należy jednak przesadzać, gdyż urządzenie może przestać rozróżniać rzeczywiste odbicie od zakłóceń.

Rys. 2





Rys. 3



W urządzeniu zaimplementowano bardzo prosty automat. Ma on trzy wspomniane wcześniej stany (*HALT*, *WAIT*, *WORK*). Poszczególne stany są reprezentowane przez instrukcje *case* wewnątrz struktury *switch*. Diagram tego automatu pokazano już na rysunku 2. Widać na nim, że przejścia pomiędzy stanami odbywają się poprzez krótkie bądź długie naciśnięcie S1. Wybór nowego stanu polega na zapisaniu odpowiedniej wartości do zmiennej *automat*. Nowy stan staje się aktywny przy kolejnej iteracji głównej pętli *while*, której kod przedstawiono na **listingu 2**.

Pierwszy stan (*STATE_HALT*) odpowiada za przygotowanie procesora do przejścia w tryb uśpienia. Wymaga to wybrania trybu *power-down* oraz ustawienie bitu *SE* w rejestrze *MCUCR*. Przy okazji wyłączne są dioda LED, buzer oraz dioda IR, aby nie zdarzyło się, że pozostaną włączone i spowodują nadmierny pobór prądu. Opóźnienie wynoszące 500ms zostało dodane, aby zapobiec niepożądanemu wybudzeniu procesora na skutek drgań zestyków. Włączenie zewnętrznego przerwania jest konieczne, aby można było wznowić pracę mikrokontrolera. Przerwanie to musi być zgłaszane poziomem niskim (nie zboczem), gdyż tylko takie umożliwia wybudzenie procesora. Ostatnim krokiem niezbędnym do przejścia w stan wstrzymania jest wydanie instrukcji *sleep* w asemblerze. Oprogramowanie napisano w C++, dlatego wykorzystano tutaj pewien trik umożliwiający wstawienie kodu w języku niskiego poziomu: *asm(„sleep”);*

W drugim stanie (*STATE_WAIT*) automatu testowany jest tor podczerwieni. Po stwierdzeniu, że występuje w nim odbicie, dioda LED jest włączana na stałe. W przypadku jego braku następuje jej miganie. Jest ono zrealizowane w sposób pośredni, tzn. poprzez zwiększanie wartości zmiennej *delay* w czasie każdej iteracji pętli *while*. Migotanie zapewnia kopiowanie jednego z bitów tej zmiennej. Można było to zrealizować prościej, z wykorzystaniem instrukcji *_delay_ms*, ale w takim przypadku powstawałoby znaczące opóźnienie w reakcji na naciśnięcie przycisku i badanie toru IR odbywałoby się rzadziej. Zastosowane rozwiązanie jest pozbawione tej wady, gdyż procesor nie zostaje wstrzymany na dłuższy czas.

Ostatni stan automatu (*STATE_WORK*) jest właściwym procesem monitorowania obiektu. Krótkie mignięcia diody są zrealizowane na podobnej zasadzie, tzn. wykorzystywana jest tu zmienna *delay*. Po stwierdzeniu, że wiązka nie ulega odbiciu, uruchamiany jest alarm. Trwa on minimalnie 500ms i jest ponawiany przy każdym braku odbicia.

Osobnego komentarza wymaga sposób obsługi toru podczerwieni. Opiera się on na kolejnej zmiennej, tym razem jest to *radar*. Po wykryciu odbicia następuje przypisanie do tej zmiennej wartości stałej *IR_FILTER*, natomiast przy braku odbicia jest ona zmniejszana o jeden, aż do zera. Rozwiązanie to stanowi bardzo prosty filtr, który zapobiega chwilowym zanikom sygnału. Może się zdarzyć tak, że tor podczerwieni będzie pracował w warunkach granicznych i czasami sygnał po prostu nie zostanie zarejestrowany (np. na skutek zakłóceń). W normalnej sytuacji spowodowałoby to od razu włączenie alarmu, ale dzięki zmniejszaniu wartości zmiennej *radar* nie nastąpi natychmiastowe włączenie alarmu. Brak odbicia musiałby wystąpić *IR_FILTER* razy pod rząd. W związku z tym stan toru IR jest przechowywany w zmiennej *radar* – kiedy osiągnie ona wartość zero można uznać, że odbicie nie występuje. Dopiero wtedy włączany jest alarm.

Na koniec mała sztuczka. Jak zaoszczędzić 0,2mA w trybie uśpienia procesora? Wyłączyć przetwornik ADC. Niby oczywiste, ale przez dłuższy czas jakoś nie mogłem wpaść na to, że tak prosta rzecz może powodować tak koszmarny wzrost poboru prądu. W programie przetwornik jest wyłączany automatycznie, zaraz po zakończeniu pomiaru. Warto o tym pamiętać, budując energooszczędne aplikacje.

Wykaz elementów

R1	430kΩ 2012 (0805)
R2	100Ω 2012 (0805)
R3,R4	1kΩ 2012 (0805)
PR1	1kΩ PR
C1	100µF tantalowy SMD
C2-C4	100nF 2012 (0805)
D2	IRED
D3	LED czerwona
T2	fototranzystor (IR)
U1	ATtiny44V (SO14)
BUZ1	buzer 3V
JP1	goldpin x 5

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-2910.

Montaż i uruchomienie

Płytka drukowana (**rysunek 3**) miała być przystosowana do obudowy KM90, jednakże na koniec okazało się, że nie uda się tam zmieścić dwóch baterii AAA w pojemniku. Czytelnicy mogą poszukać alternatywnego sposobu zasilania urządzenia, np. baterią CR2032 bądź jakimś akumulatorem 3,6V wymontowanym z telefonu komórkowego. Obudowa nie ma dedykowanych punktów mocujących dla płytki drukowanej, ale problem łatwo obejść, przyklejając dwie plastikowe tulejki dystansowe klejem superglue. Dioda IR oraz fototranzystor mogą zostać umieszczone w specjalnych, plastikowych oprawkach dostępnych w sklepach elektronicznych.

Krótkie naciśnięcie przycisku S1 powoduje natychmiastowe przejście procesora w stan pracy aktywnej i wzrost poboru prądu. Warto unikać przypadkowego uruchomienia urządzenia i w jakiś sposób zabezpieczyć przycisk, np. tak, aby nie wystawał z obudowy. Wymaga to co prawda jakiegoś narzędzia do włączenia i zmiany trybu pracy (np. szpilki), ale zapobiega również przypadkowemu włączeniu.

Do sygnalizowania alarmu wykorzystałem buzer z wbudowanym generatorem. Jego głośność wydała mi się wystarczająca do tego celu. Element ten został bezpośrednio wlutowany w płytkę.

Montaż najlepiej rozpocząć od wlutowania podzespołów wykonanych w technologii SMD pozostawiając elementy przewlekane na sam koniec.

Możliwości zmian

Czytelnikom pozostaje możliwość wprowadzenia drobnych modyfikacji w kodzie takich jak zmiana wartości stałej *IR_FILTER*, modyfikacja czasu trwania alarmu czy ustawienie czułości odbiornika poprzez modyfikację parametru *IR_SENSOR_DELTA*. Bardziej doświadczone osoby mogą pokusić się o wprowadzenie własnych ulepszeń, aby lepiej dostosować urządzenie do indywidualnych oczekiwań.

Jakub Borzdyński
jakub.borzdynski@elportal.pl