Programowe dekodowanie sygnałów zdalnego sterowania

Większość współcześnie produkowanych elektronicznych urządzeń powszechnego użytku jest wyposażona w zdalne sterowanie, czyli tzw. „pilota". Dostępność pilotów zamiennych oraz ich niska cena skłania do wykorzystania tego „dobrodziejstwa" we własnych projektach. Celem niniejszego artykułu jest przybliżenie Czytelnikom sposobu dekodowania - za pomocą dowolnego mikrokontrolera - sygnałów zdalnego sterowania wysyłanych przez większość obecnie stosowanych pilotów.

Na początek trochę historii

W pierwszych systemach bezprzewodowego zdalnego sterowania wykorzystywano do przekazywania informacji o naciśnięciu klawiszy sygnały wielocz̨stotliwościowe. Oznaczało to, że każdemu klawiszowi pilota przyporządkowana była fala prostokątna o określonej częstotliwości, która służyła do sterowania przetwornika piezoceramicznego lub do modulacji światła diody LED emitującej światło podczerwone. System ten został dość szybko zastąpiony przez bardziej zaawansowane systemy cyfrowej transmisji danych, ze względu na dużą komplikacje układu odbiorczego oraz niewielką liczbę komend możliwych do przekazania. Zaniechano również stosowania ultradźwięków jako medium transmisyjnego ze względu na dużą podatność na zakłócenia oraz interferencje z sygnałem odbitym od ścian pomieszczenia itp.

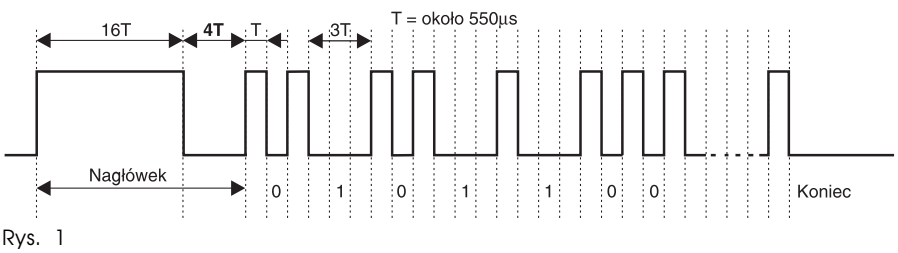
W wyniku ewolucji powstało kilka standardów kodowania sygnałów zdalnego sterowania, a do ich przekazywania stosuje się nadal promieniowanie podczerwone.

Zasada działania toru transmisyjnego

Budowa współczesnego pilota jest bardzo prosta. Składa się on z klawiatury, mikrokontrolera lub specjalizowanego układu sterującego, wzmacniacza prądowego oraz jednej lub kilku diod emitujących światło podczerwone. Mikrokontroler realizuje wszystkie funkcje związane z identyfikacją naciśniętego klawisza, przypisania mu odpowiedniego kodu oraz wysłania odpowiedniego ciągu impulsów na wyjście sterujące diodą świecącą. Dla zwiększenia odporności na zakłócenia stosuje się dodatkowo modulację sygnałem cyfrowym danych kodu sygnał o określonej częstotliwości nośnej, która ułatwia odróżnienie właściwego sygnału pilota od zakłóceń wywołanych oświetleniem odbiornika przez światło słoneczne lub sztuczne. W związku z tym dioda świecąca w pilocie nie świeci ciągłym światłem, tylko „mruga" z częstotliwością od 35 do 40 kHz.

Odbiornik składa się z filtru podczerwieni, za którym umieszczona jest fotodioda przekształcająca padające promieniowanie świetlne na sygnał elektryczny. Po odpowiednim wzmocnieniu we wzmacniaczu wejściowym, sygnał jest przepuszczony przez filtr pasmowo-przepustowy zestrojony na częstotliwość nośną (35...40 kHz), a następnie podawany na detektor z przerzutnikiem Schmitta. Wzmacniacz i filtr objęty jest pętlą automatycznej regulacji wzmocnienia, dzięki czemu ustalane jest odpowiednie wzmocnienie układu w zależności od natężenia sygnału promieniowania podczerwonego i poziomu zakłóceń. Na wyjściu detektora uzyskuje się ciąg bitów odpowiadający kodowi naciśniętego klawisza.

Zazwyczaj odbiornik jest wykonany jako pojedynczy element, zamknięty w trójkońcówkowej obudowie wykonanej z tworzywa przepuszczającego promienie podczerwone. Aby ułatwić współpracę odbiornika z mikrokontrolerem, sygnał wyjściowy odbiornika ma postać zanegowaną. Oznacza to, że w stanie spoczynkowym na wyjściu odbiornika występuje poziom wysoki, a pojawienie się sygnału (impulsu) powoduje występowanie na nim poziomu niskiego (zbocza opadającego).



Rys. 1

Sposoby kodowania danych

Każde naciśnięcie klawisza w pilocie powoduje wysłanie paczki impulsów identyfikującej urządzenie, do którego dany sygnał jest przeznaczony (adres) oraz kod naciśniętego klawisza. Ta paczka impulsów składa się z kilkunastu lub kilkudziesięciu bitów zakodowanych w sposób umożliwiający jak najlepsze odróżnienie bitu o wartości „0" od „1" - czasami uzupełnionych o bity startu, stopu i sygnał „rozbiegowy" poprzedzający transmisję. Bity startu mają na celu odpowiednie wysterowanie układu ARW w odbiorniku i dostosowanie wzmocnienia do bieżącego poziomu sygnału IR. Ponad 95% stosowanych obecnie pilotów wykorzystuje jeden z trzech sposobów kodowania przesyłanych informacji. Przedstawiamy je poniżej.

Space Coding

Pierwszy z nich, zwany *space coding*, polega na modyfikacji czasu przerwy pomiędzy przesyłanymi impulsami (**rys. 1**). W tym systemie czas trwania impulsu jest stały - nazwijmy go T. Jeśli dwa kolejne impulsy przedzielone są przerwą o czasie trwania równym T, to przesyłany bit ma wartość „0", a jeśli przerwa ta wynosi 3T, to bit ma wartość „1" Ostatni bit zakończony jest impulsem o czasie trwania T. Wartość czasu T wynosi od 400 do 600 µs. Transmisja jednego rozkazu poprzedzona jest

List. 1.

```
// kwarc 7.372MHz
#define IR_PORT  PORTB
#define IR_BIT   PB0                    // PB0 to wejście z odbiornika

unsigned int get_ir(unsigned char std)    // parametr std określa standard (0-4)
{
   unsigned char i, T2, T4, time, tmp = 0;
   unsigned int code;

   code = 0;
   timer0_source(CK256);                // prescaler timera 0 na ok. 32us
   timer0_start();                      // uruchom timer
   loop_until_bit_is_set(IR_PORT-2, IR_BIT); // pomiń nagłówek

   if(std < 2)                          // standard REC80 lub NEC80
   {
      timer0_start();
      while (bit_is_set(IR_PORT-2, IR_BIT))
      {
         T2 = inp(TCNT0);
         if (T2 >= 140)                 // maksymalny czas oczekiwania ok. 5ms
            return 0;                   // powrót z błędem
      }

      // Pomiar czasu T
      timer0_start();                   // Uruchom timer
      loop_until_bit_is_set(IR_PORT-2, IR_BIT);
      T2 = inp(TCNT0);                  // Odczytaj czas
      T2 = T2 * 2;                      // Punkt podzialu (T lub 3T)
      T4 = T2 * 2;                      // maksymalny czas oczekiwania na bit(4T)

      // pętla dekodowania kolejnych bitów
      // 48 bitów dla REC80, 32 dla NEC80
      for (i = 0; i < ((std == 0) ? 48 : 32); i++)
      {
         timer0_start();                // uruchom timer
         while(1)
         {
            time = inp(TCNT0);
            if (time > T4)
               return 0;                // przekroczenie czasu bitu

            // pomiar czasu trwania "0"
            if (bit_is_clear(IR_PORT-2, IR_BIT)) // logika ujemna!
            {
               tmp <<= 1;               // przesuń wynik o 1 bit
               if (time >= T2)          // jeśli czas większy od 2T
                  tmp++;                // bit jest jedynką (ustaw LSB)
               break;                   // wyjdź z pętli
            }
         }

         // zapamiętanie wyniku
         if (std == 0)                  // REC80
         {
            if( i == 39)                // starszy bajt
               code = (u16)tmp << 8;
            if( i == 47)                // młodszy bajt
               code += tmp;
         }
         else                           // NEC80
         {
            if( i == 15)                // bardziej znaczący bajt
               code = (u16)tmp << 8;
            if( i == 31)                // mniej znaczący bajt
               code += tmp;
         }

         // zaczekaj na nastepne "0"
         loop_until_bit_is_set(IR_PORT - 2, IR_BIT);
      }
      return (code);                    // koniec odbioru standardu SONY
   }

   else if(std < 4)                     // Standard SONY
   {
      if (inp(TCNT0) <= 60)             // Nagłówek ?
         return 0;                      // Nie - wróć z błędem

      // pętla dekodowania kolejnych bitów
      // 12 lub 15 bitów do odczytania
      for(i=0; i < ((std == 3) ? 11 : 14); i++)
      {
         tmp = 0x01;
         // poczekaj na początek impulsu
         while (bit_is_set(IR_PORT-2, IR_BIT))
         {
            T2 = inp(TCNT0);
            if (T2 >= 140)              // przekroczenie czasu bitu ?
               return 0;                // błąd
         }
         timer0_start();                // pomiar czasu trwania impulsu
```

nagłówkiem składającym się z sygnału trwającego 16T oraz przerwy trwającej 4T. Standard ten stosowany jest głównie przez firmę Panasonic, ale jest bardzo chętnie wykorzystywany przez wiele innych firm. Istnieją dwie odmiany tego standardu: REC80, w którym przesyłane jest 48 bitów danych, oraz NEC80, w którym przesyła się 32 bity.

Pulse Coding

Kolejny sposób kodowania - zwany *pulse coding* - polega na modyfikacji szerokości emitowanego impulsu (**rys. 2**). Ten system kodowania jest stosowany głównie przez firmę Sony. W tym sposobie „1“ logicznej przyporządkowano impuls trwający 1,2 ms, a „0“ logicznemu impuls o czasie 0,6 ms. Poszczególne impulsy przedzielone są przerwą o stałej długości - 0,6 ms. Nagłówek transmisji składa się z impulsu trwającego 2,4 ms i przerwy o długości 0,6 ms. Standard ten występuje również w dwóch odmianach, w których przesyłane jest 12 lub 15 bitów kodu, począwszy od bitu najmniej znaczącego. Transmisja jednego kodu w tym standardzie trwa ok. 45 ms.

Shift Coding

Trzeci sposób kodowania sygnałów zdalnego sterowania - RC-5 - jest stosowany przez firmę Philips. Po naciśnięciu klawisza pilota jest generowane 14-bitowe słowo kodowe zawierające 2 bity startowe, bit świadczący o przytrzymaniu klawisza, 5-bitowy adres urządzenia i 6-bitowy kod przesyłanej komendy (**rys. 3**). Czas trwania jednego bitu wynosi 1,778 ms, a transmisji kompletnego słowa kodowego 25ms. Odstęp między kolejnymi słowami kodu wynosi 114 ms. Bity ramki są kodowane bifazowo (inaczej *Shift Coding*) - jedynka logiczna składa się kolejno z przerwy i z impulsu o czasach trwania równych połowie czasu trwania bitu, a zero logiczne odwrotnie - czyli z impulsu oraz przerwy. Inaczej mówiąc, przy „1“ mamy narastające zbocze sygnału w połowie czasu trwania bitu, a przy „0“ - opadające.

Dekodowanie

Sygnał wyjściowy z odbiornika podczerwieni najłatwiej jest dekodować za pomocą odpowiednio oprogramowanego mikrokontrolera. Wystarczy wprowadzić ten sygnał bezpośrednio na dowolną linię portu I/O skonfigurowanego jako wejście, a całą „robotę“ wykona zawarty w mikrokontrolerze program. Czasami dobrze jest, gdy linia wejściowa mikrokontrolera ma możliwość wygenerowania przerwania w celu powiadomienia mikrokontrolera o początku transmisji, lecz nie jest to niezbędne. Do dekodowania sygnałów każdego z opisanych standardów jest potrzebna inna procedura, wiec musimy z góry określić, jaki sygnał będziemy dekodować lub umieścić w pamięci mikrokontrolera wszystkie procedury dekodowania i włączyć w program sterujący pracą mikrokontrolera procedu-

List. 1 - cd.

```
          while (bit_is_clear(IR_PORT-2, IR_BIT))
          {
              T2 = inp(TCNT0);
              if (T2 >= 140)           // przekroczenie czasu bitu
                  return 0;            // błąd
          }
          if (inp(TCNT0) >= 25)        // czas impulsu większy niż 1ms ?
              code += ((u16)tmp << i); // ustaw odpowiedni bit wyniku
      }
      return (code);                   // koniec odbioru
   }

   else                                // std=4 czyli RC-5;
   {
      for(i=0; i<13; i++)              // pozostało 13 bitów
      {
          if(bit_is_clear(IR_PORT-2, IR_BIT) )
              T2 = 0;                  // aktualnie jest 1
          else
              T2 = 1;                  // aktualnie jest 0

          timer0_start();              // uruchom timer
          while(1)
          {
              time=inp(TCNT0);
              if(time > 0x21)          // przekroczenie czasu bitu ?
                  return 0;            // błąd
              // narastające zbocze w połowie bitu ?
              if(bit_is_clear(IR_PORT-2, IR_BIT) && (T2==1) )
              {
                  tmp <<= 1;           // tak - przesuń wynik
                  tmp++;               // i zapisz "1"
                  break;
              }
              // opadające zbocze w połowie bitu ?
              else if(bit_is_set(IR_PORT-2, IR_BIT) && (T2==0) )
              {
                  tmp <<= 1;           // tak - przesuń wynik
                  break;               // i zapisz "0"
              }
          }

          // zapamiętanie adresu urządzenia
          if(i == 6)
          {                            // zapisz adres i
              code = (tmp & 0x5f) << 8;   // obetnij troggle bit
              tmp=0;                   // zeruj bajt odbioru
          }

          timer0_start();              // opóźnienie o 3/4 czasu bitu
          while(1)
          {
              time=inp(TCNT0);
              if(time > 0x21)
                  break;
          }
      }
      code += tmp;                     // zapamiętanie kodu komendy
      return(code);                    // koniec odbioru standardu RC-5
   }
}
```

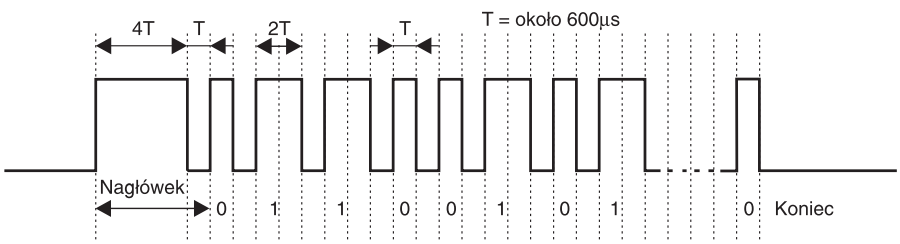
rę rozpoznania standardu. Do odmierzania określonych odcinków czasu najlepiej jest wykorzystać sprzętowy timer. Na **list. 1** przedstawiono uniwersalną funkcję dekodowania sygnałów poszczególnych standardów we wszystkich pięciu wersjach napisaną w języku C, przeznaczoną dla mikrokontrolerów AVR (do skompilowania bezpłatnym kompilatorem AVR-GCC). Jako parametr należy podać numer standardu (0=REC80, 1=NEC80, 2=SONY15, 3=SONY12, 4=RC5). Funkcja zwraca dwubajtowy kod naciśniętego klawisza lub wartość 0, jeśli wystąpił błąd odbioru. Dla niektórych standardów jest zwracany tylko fragment słowa kodowego, lecz wystarcza to do jednoznacznej identyfikacji naciśniętego klawisza.

Na początku inicjowane jest kilka zmiennych oraz uruchamiany timer. Konfiguracja preskalera powoduje, że timer zwiększa swoją zawartość co około 32 µs. Następnie trzeba odczekać do końca czasu trwania impulsu. Po pojawieniu się na wejściu poziomu wysokiego, w zależności od wybranego standardu, sterowanie zostaje przekazane do jednego z trzech fragmentów.

Jeśli zmienna *std* jest równa 0 lub 1, to będziemy dekodować sygnał w formacie *Space*. Najpierw omijana jest pierwsza przerwa i jest sprawdzane, czy nie jest zbyt długa. Następnie zerowany jest licznik timera i rozpoczyna się pomiar czasu trwania impulsu. Zmierzona wartość czasu po pomnożeniu przez 2 służy do rozróżnienia „0“ trwającego T i „1“ trwającej 3T. Na 4T zostaje ustawiony maksymalny czas trwania przerwy pomiędzy impulsami. Następnie rozpoczyna się pętla odbioru 32 lub 48 bitów danych, w której mierzony jest czas trwania przerwy pomiędzy impulsami. Jeśli jest on krótszy od 2T, to odczytany bit ma wartość „0“, jeśli pomiędzy 2T a 4T, to odczytany bit ma wartość „1“, a jeśli powyżej 4T, to generowany jest błąd i funkcja *get_ir* zwraca wartość 0. Następnie sprawdzany jest numer odbieranego bitu i w odpowiednim momencie, w zależności od tego, czy odbieramy 32, czy 48 bitów, kolejno do bardziej i mniej znaczącego bajtu zmiennej *code* wpisywana jest zawartość tymczasowego rejestru odbioru. Następnie odczekuje się do końca czasu trwania impulsu i rozpoczyna się kolejny obieg pętli. Po odczytaniu wszystkich bitów zwracana jest wartość zmiennej *code* i funkcja kończy działanie.

Jeśli zmienna *std* jest równa 2 lub 3, to będziemy dekodować sygnały w formacie *Pulse*. Najpierw sprawdzane jest, czy pierwszy impuls trwał dłużej niż 2 milisekundy, co oznacza, że był to prawidłowy nagłówek. Następnie rozpoczyna się pętla odbioru kolejnych 11 lub 14 bitów. W zmiennej *tmp* ustawiamy „1“ na najmniej znaczącej pozycji. Będzie ona służyła jako maska numeru odbieranego bitu. Następnie należy odczekać do końca trwania przerwy, sprawdzając przy okazji, czy nie była zbyt długa. Następnie zerowany jest timer i rozpoczyna się pomiar czasu trwania impulsu. Jeśli zmierzony czas jest dłuższy od 1 ms, to odebrany bit jest równy „1“, jeśli krótszy, to „0“. Przekroczenie czasu 4,5 ms traktowane jest jako błąd odbioru. Jeśli bit był jedynką, to na „1“ ustawiany jest również odpowiedni bit zmiennej *code*. Na tym kończy się jeden obieg pętli i rozpoczynamy odbiór kolejnego bitu. Po odczytaniu wszystkich bitów zwracana jest wartość zmiennej *code* i funkcja kończy działanie.

Jeśli zmienna *std* jest równa 4, to będziemy dekodować sygnał w formacie *Shift*, czyli RC-5. Ponieważ pierwszy impuls został pominięty na początku, dekodowanych jest 13 pozostałych bitów. Na początku pętli sprawdzany jest i zapamiętywany w zmiennej *T2* aktualny stan wejścia. Następnie zerowany



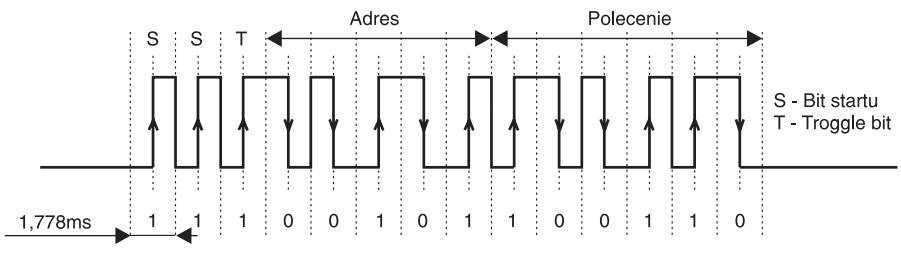
Rys. 2

List. 2.
```
unsigned char ir_active(void)
{
   unsigned char i = 50;           // liczba kolejnych odczytów stanu wejścia
   while (i-- != 0)
   {
       if (bit_is_clear(IR_PORT-2, IR_BIT))   // testuj wejście
           return 1;               // jest stan niski
   }
   return 0;                       // no nie tym razem ☺
}

...
// gdzieś w głównej pętli programu
if (ir_active())                   // jeśli stwierdzono aktywność
   if (code = get_ir(standard));   // odbierz kod i jeśli nie ma błędu
   {
       ...                         // tu jest reakcja na pilota
   }
...
```

List. 3.
```
SIGNAL(SIG_ INTERRUPT0)
{
   unsigned int temp;
   temp = get_ir(standard);
   if (temp)
   {
       code = temp;      // code - globalna zmienna zawierająca kod klawisza
       ir_flag = 1;      // ir_flag - flaga prawidłowego odebrania transmisji
   }                     // zerowana po obsłużeniu wyniku w programie głównym
}

...
// gdzieś w głównej pętli programu
if (ir_flag)             // jeśli stwierdzono aktywność
{
   ...                   // tu jest reakcja na pilota
}
...
```

List. 4.
```
unsigned char recognize_std(void)
{
   unsigned char stand = 0;     // numer standardu
   unsigned char i=4;           // licznik prób

   while(1)
   {
       if (ir_active())         // jeśli wykryto sygnał
       {
           if (get_ir(stand))   // jeśli nie ma błędu
               return (stand);  // znaleziono właściwy standard -> koniec
           else
           {
               if(--i == 0)     // zmniejsz licznik prób i jeśli =0
               {
                   stand = (stand+1) % 5;   // sprawdź kolejny standard
                   i=4;         // ustaw od nowa licznik prób
               }
           }
       }
   }

}
```

jest timer i rozpoczyna się odbiór jednego bitu. Jeśli w czasie 1 ms nastąpi zmiana poziomu na wejściu odbiornika, to w zależności od poprzedniego stanu mamy narastające zbocze w połowie bitu, które oznacza, że bit ma wartość „0“ lub opadające zbocze oznaczające bit równy „1“. Zapisujemy go do najmniej znaczącego bitu zmiennej *tmp* i przesuwamy ją w lewo. Następnie odczekujemy około 0,5 czasu trwania jednego bitu, aby można było określić kierunek kolejnego zbocza występującego w połowie bitu. Po odebraniu szóstego bitu, do bardziej znaczącego bajtu zmiennej *code* zapisywany jest kod urządzenia (po usunięciu bitu świadczącego o przytrzymaniu klawisza, który zmienia wartość po każdym naciśnięciu klawisza) oraz zerowana jest zmienna *tmp*. Po odebraniu ostatniego bitu, w mniej znaczącym bajcie wyniku zapamiętywany jest kod komendy i funkcja zwraca wartość zmiennej *code*, kończąc działanie.



Rys. 3

Funkcje pomocnicze

Funkcje *get_ir* należy wywołać w momencie wykrycia „0“ logicznego na wejściu z odbiornika, czyli w momencie pojawienia się pierwszego impulsu transmisji. Pamiętajmy, że sygnał wyjściowy odbiornika jest zanegowany, więc obecności sygnału odpowiada poziom niski na wejściu mikrokontrolera. Można wykryć go dwoma sposobami:
- poprzez okresowe sprawdzanie stanu linii procesora,
- z wykorzystaniem do tego celu przerwań.

Pierwszy sposób - pokazany na **list. 2** - jest łatwiejszy, lecz zawsze zużywa nieco czasu procesora. Funkcja *ir_active* zwraca wartość „0“, jeśli nie wykryto sygnału IR lub 1 w momencie wykrycia poziomu niskiego (czyli impulsu). Drugi sposób - pokazany na **list. 3** - nie ma wady poprzedniego rozwiązania, lecz również nie jest idealny. A to dlatego, że jeśli odbiornik zostanie oświetlony silnym modulowanym światłem (np. bezpośrednio świetlówką kompaktową), to na jego wyjściu może pojawić się seria przypadkowych impulsów generujących kolejne przerwania i próby odczytu sygnału pilota, co spowoduje jeszcze większe spowolnienie programu głównego. Niestety, nic nie jest idealne i czasami trzeba wybierać mniejsze zło.

Na **list. 4** pokazano sposób automatycznego rozpoznania standardu, w którym nadaje pilot. Wystarczy, że po uruchomieniu tej procedury kilkakrotnie naciśniemy dowolny klawisz pilota, a jego standard zostanie zwrócony jako wynik funkcji *recognize_std* (można go później zapisać np. w EEPROM-ie). Wykorzystujemy go również jako parametr wywołania funkcji *get_ir*.

Na CD-EP12/2002B oraz w Internecie (*www.ep.com.pl*) publikujemy te same procedury (*get_ir* oraz *recognize_std*) napisane w asemblerze procesora AVR, z przeznaczeniem na małe procesory bez wewnętrznego RAM-u (np. AT90S1200 lub ATtiny). Procedura rozpoznania standardu zapisuje jego numer w wewnętrznej pamięci EEPROM procesora dla wykorzystania w programie głównym.

Mam nadzieję, że przedstawiony opis i przykłady ułatwią Czytelnikom użycie pilota we własnych projektach i pomogą w napisaniu swojej wersji procedur dekodujących przy wykorzystaniu do tego celu mikrokontrolera innego niż AVR.

Romuald Biały