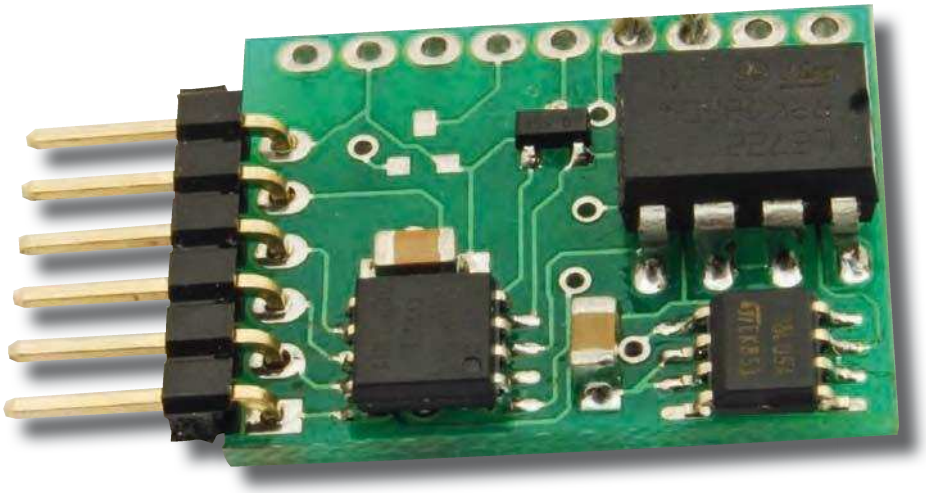
Dekoder DCC

Sterowanie makietą kolejową

Spośród wielu tematów prezentowanych w EP, niesłabnącym zainteresowaniem cieszą się te związane z modelarstwem. Faktycznie, współczesna elektronika stwarza miłośnikom budowy wszelkich modeli ogromne możliwości. W poprzednim numerze opublikowaliśmy opis układu Samoczynnej Blokady Liniowej zapobiegającego kolizjom na makiecie kolejowej. Teraz opisujemy kolejny komponent, umożliwiający sterowanie ruchem na makiecie. W kolejnych numerach opublikujemy pulpit sterowniczy, jak również układ generatora dźwięków lokomotywy.
Rekomendacje: *modelarze kolejowi.*



Modelarzom kolejowym nie trzeba tłumaczyć co to jest dekoder DCC i do czego służy. Niewtajemniczonym powiem, że umożliwia on sterowanie lokomotywami na makiecie. Każda z nich może jechać w innym kierunku i z inną prędkością. Ponadto może realizować dodatkowe funkcje, takie jak sterowanie oświetleniem, rozsprzęganie, generowanie dymu czy dźwięków.

W Internecie dostępnych jest wiele rozwiązań dekoderów. Konstrukcja większości z nich oparta jest o mikrokontrolery PIC-16F84A. Zbudowałem kilka takich dekoderów, ale niestety nie spełniały one moich wymagań. Największą ich wadą był brak funkcji ABC (zatrzymanie lokomotywy przed semaforem). Spowodowane jest to brakiem przetwornika AC w PIC16F84A. Ponadto procesor w obudowie DIP jest stosunkowo duży i wymaga zewnętrznego rezonatora, co także wpływa na gabaryty dekodera. Można oczywiście zastosować procesor w obudowie SMD, ale pojawiają się kłopoty ze zmianą programu w już zmontowanym urządzeniu. Z takich powodów postanowiłem więc zbudować dekoder w oparciu o procesor AVR.

AVR-y dostępne są również w małej, mającej osiem wyprowadzeń obudowie, co umożliwia miniaturyzację dekodera. Ponadto mają one wbudowany przetwornik A/C, co daje możliwość implementacji funkcji ABC. Wbudowany generator PWM ułatwia sterowanie silnikiem, a pamięć EEPROM umożliwia zapamiętanie konfiguracji i jej zmianę bezpośrednio na makiecie. Ponadto AVR-y mogą pracować bez zewnętrznego generatora kwarcowego.

Mój wybór padł na ATTiny85. Ten mikrokontroler ma małe wymiary, niską cenę zakupu. Do jego programowania można zastosować tani debuger „AVR Dragon". Ponadto, jego zaletami są stosunkowo duża pamięć programu (8 KB) oraz danych (1 KB), co umożliwiło napisanie programu w języku C. Cały kod programu jest dostępny na płycie CD EP9/2009B oraz na stronie *www.kolejki-h0.pl*.

Dekoder realizuje następujące funkcje:
- Sterowanie silnikiem, to jest kierunkiem i prędkością obrotów w 28 krokach; prąd stały 0,8 A.
- Ustawienie minimalnego napięcia zasilania silnika, przy którym lokomotywa zaczyna się poruszać.
- Ustawienie maksymalnego napięcia zasilania silnika.

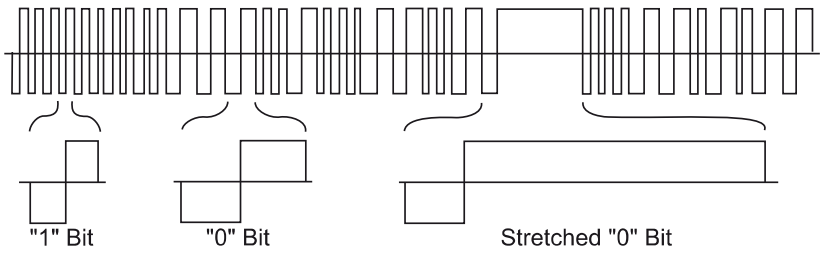
AVT–5201

W ofercie AVT:
AVT–5201A – płytka drukowana

PODSTAWOWE PARAMETRY
• Zgodny ze specyfikacją DCC
• Płytka dwustronna 25×20 mm
• Mikrokontroler ATtiny85
• Zasilanie do 20 VDC
• Sterowanie silnikiem lokomotywy, oświetleniem i innymi, dodatkowymi funkcjami
• Uniwersalny: możliwość zastosowania do sterowania semaforem, przejazdem kolejowym itp.

PROJEKTY POKREWNE wymienione artykuły są w całości dostępne na CD

Tytuł artykułu	Nr EP/EdW	Kit
Samoczynna Blokada Liniowa	EP8/2009	AVT–5198



Rys. 1. Ramka transmisji DCC

- Ustawienie częstotliwości PWM zasilania silnika.
- Ustawienie czasu rozpędzania.
- Ustawienie czasu hamowania.
- Sterowanie jedną funkcją (np. oświetleniem).
- ABC.
- Konfigurowanie dekodera w lokomotywie.
- Jazda analogowa.
- Złącze NMRA651.

Transmisja sygnału w systemie DCC

Zanim rozpocząłem projektowanie dekodera przejrzałem specyfikację NMRA. Postaram się trochę przybliżyć sposób transmisji sygnału w systemie DCC.

Napięcie na szynach jest napięciem przemiennym (jak w RS485 czy 422). Zmiany sygnału następują w różnym czasie, zależnie od tego czy transmitowane jest 0, czy 1. Warto wiedzieć, że czasy trwania polaryzacji dodatniej i ujemnej w transmitowanym bicie są sobie równe i w wyniku tego średnie napięcie na torach wynosi 0 V. Jedynka reprezentowana jest przez impulsy o czasie trwania 55...61 µs, natomiast zero przez impulsy o czasie trwania 91 µs...1 ms. Typowo jedynka to impuls o maksymalnym czasie trwania 120 µs, a do czego służą dłuższe impulsy napiszę dalej. W moim układzie jedynka to impuls o okresie 114 µs i wypełnieniu 50%, a zero to impuls o okresie 200 µs i wypełnieniu 50%. Jak łatwo wywnioskować, aby poprawnie zdekodować sygnał DCC, wystarczy tylko mierzyć czas trwania impulsów na jednej szynie (tylko dodatnich lub tylko ujemnych). Kolejną zaletą takiej transmisji jest fakt, że niezależnie od tego, jak postawimy lokomotywę, sygnał zawsze zostanie poprawnie zdekodowany. Polaryzacja sygnału na szynach nie ma znaczenia, ponieważ jest on symetryczny. Ramkę transmisji DCC przedstawiono na **rys. 1**.

Czasy połowiczne transmitowanego bitu są sobie równe, więc średnie napięcie na torach jest równe 0 V. Jest jednak pewne odstępstwo od tej reguły. Jedna z połówek bitu 0 może zostać wydłużona nawet do 1 ms. Spowoduje to, wzrost napięcia średniego nawet do 90% amplitudy sygnału DCC. Zależnie od tego czy przedłużymy dodatnią, czy ujemną połówkę, napięcie będzie się zmieniało do –90% do +90% amplitudy. Pytanie czemu to służy? Umożliwia to sterowanie jedną lokomotywą analogową, bez zamontowanego dekodera DCC. Trzeba jednak uważać na maksymalna amplitudę sygnału DCC. Najczęściej sięga ona 20 V, co może spowodować uszkodzenie silnika w lokomotywie bez dekodera. Trzeba pamiętać, że przez silnik takiej lokomotywy płynie maksymalny prąd nawet wtedy, gdy stoi nieruchomo na torze. Sterując taką lokomotywą należy zmniejszyć amplitudę sygnału na bosterze do 12 V, maksymalnie 14 V.

Ramka DCC składa się z preambuły, na którą przypada 14 bitów o wartości 1 (dekoder akceptuje preambułę o minimalnej długości 12 bitów). Po preambule nadawany jest bit „0” oznaczający początek transmisji. Następnie nadawany jest adres lokomotywy, po czym znowu „0”. Kolejny bajt to komenda, której transmisję kończy „0”. Ostatnim bajtem jest suma kontrolna EXOR i bit o wartości „1”. Dopuszcza się, aby bit końca pakietu był jednocześnie pierwszym bitem preambuły. Przykładowa ramka transmitująca prędkość, kierunek jazdy i sterowanie funkcją F0 może wyglądać tak:

```
111111111111 0 [AAAAAAAA] 0
01DFSSSS 0 EEEEEEEE 1
```

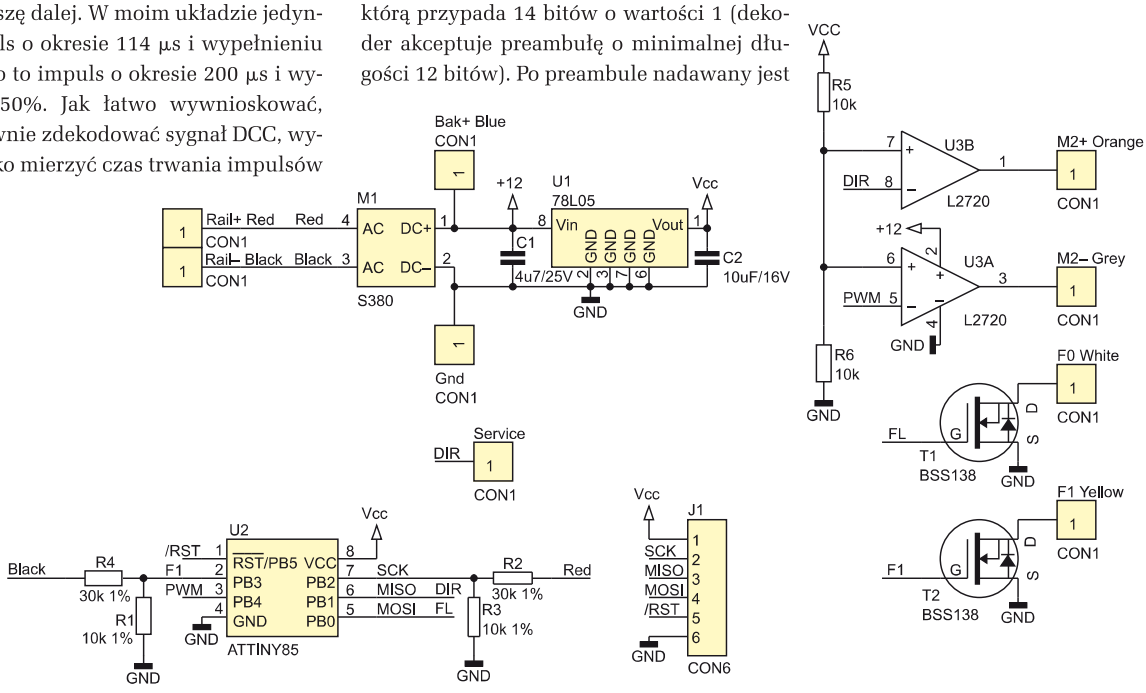
gdzie:
AAAAAAAA – adres lokomotywy
D – kierunek jazdy
F – sterowanie funkcją F0
SSSS – prędkość jazdy

Jak łatwo się zorientować, można ustawić jedną z 16 prędkości jazdy, a w praktyce 14, ponieważ „0” to stop, natomiast „1” to zatrzymanie awaryjne.

Dekoder obsługuje 28 kroków (32 kroki minus dwie kombinacje stop i 2 kombinacje zatrzymania awaryjnego). Jest to możliwe, ponieważ bit F może być używany jako 5 bit (najmłodszy) parametru prędkości jazdy. Informacje o sterowaniu oświetleniem są przesyłane w ramce sterujące wyjściami funkcyjnymi F1...F4. Osoby zainteresowane formatem innych ramek zachęcam do przeczytania dokumentacji zawartej na CD-ROM

Budowa i uwagi nt. programowania mikrokontrolera

Po krótkiej teorii czas przejść do opisu budowy dekodera. Pierwszy projekt oparty był o ATTiny85 w obudowie DIP i końcówce mocy na ICL7667 w obudowie SO-8. Próby wypadły pozytywnie, więc opracowałem kolejna wersję, tym razem na ATTiny85 w obudowie SOIC-8. Zamontowałem go w loko-



Rys. 2. Schemat ideowy dekodera DCC

WYKAZ ELEMENTÓW
Rezystory (SMD 1206)
R1, R3, R5, R6: 10 kΩ
R2, R4: 30 kΩ
Kondensatory
C1: 4,7 µF/25 V
C2: 10 µF/16 V
Półprzewodniki
U1: 78L05 (SO-8)
U2: ATTiny85 (SOIC-8)
U3: L2720 (SO-8)
M1: mostek prostowniczy S380
T1, T2: BSS138 (SOT-23)

motywie i zaczęły się kłopoty. Okazało się, że często uszkadzał się ICL7667. Działo się to jednak tylko na makiecie i nie w każdej lokomotywie. Okazało się, że podczas prób boster zasilałem z zasilacza napięciem 18 V, dzięki czemu napięcie zasilające ICL7667 nie przekraczało dopuszczalnych 15 V (3 V to suma spadków na mostku w bosterze, mostku w dekoderze oraz na układzie L298 w bosterze). Natomiast na makiecie boster zasilany był napięciem ponad 20 V, co powodowało uszkodzenie ICL7667 przy dużych prędkościach. Byłem zmuszony do zastosowania wzmacniacza na układzie L2722. Niestety, nie jest on dostępny w obudowie SMD oraz potrzebuje dwóch dodatkowych rezystorów. Schemat dekodera można zobaczyć na **rys. 2**.

Napięcie przemienne z torów jest prostowane w mostku M1 i filtrowane przez C1. Zaciski „Bak+Blue" i „GND" umożliwiają podłączenie kondensatora o dużej pojemności, rzędu 1000...2200 µF/25 V podtrzymującego zasilanie w przypadku przejazdu lokomotywy przez zwrotnicę. Bez tego kondensatora układ pracuje poprawnie, ale przy wolnej jeździe lokomotywy mogą pojawić się kłopoty. Przeprowadzając testy z wieloma lokomotywami doszedłem do wniosku, że warto zastosować jakikolwiek kondensator, nawet o pojemności 100 µF.

Silnik sterowany jest przez wzmacniacz U3 typu L2722. Funkcjami (np. oświetleniem) steruje tranzystor T1 typu BSS138. Tranzystor T2 nie jest potrzebny, jeśli dekoder wykorzystywany jest standardowo. Może on znaleźć zastosowanie, gdy do procesora wgramy program obsługi akcesoriów. Wtedy dekoder może sterować zwrotnicą elektromagnetyczną (wyjścia F0 i F1), zwrotnicą z silnikiem (wyjścia M+ i M–), 4-komorowym semaforem świetlnym (wszystkie wyjścia) lub semaforem kształtowym (wyjścia M+ i M– zabezpieczone przed przepięciami).

Napięcie do zasilania mikrokontrolera stabilizowane jest przez U1. Pracą dekodera steruje mikrokontroler U2, który można programować w systemie. Programator podłącza się do J1. Podczas użytkowania Dragona okazało się, że wbudowane w niego rezystory podciągające, uniemożliwiały poprawną pracę wejścia PB2. Dlatego po zaprogramowaniu procesora należy odłączyć programator od J1. Innym rozwiązaniem jest przejście w tryb DebugWire i zastosowanie przewodu bez podłączonego pinu 2 na złączu J1 lub jeszcze lepiej – włącznika.

Sygnał z torów trafia przez dzielniki R1–R4 i R2–R3 na wejścia mikrokontrolera. Pełnią one dwojaką funkcję: odbierają dane cyfrowe z szyn oraz mierzą napięcie na szynach. Jeśli zostanie wykryta zbyt duża asymetria, to dekoder może wyhamować lokomotywę (o ile odpowiednio skonfigurowany).

Oprogramowanie

Napisanie programu nie było proste. Pierwotnie sprawdzano stan wyprowadzeń w przerwaniu wywoływanym co 10 µs. Niestety, były problemy z rozróżnieniem „1" od „0". Jedynka powinna być akceptowana dla czasów 52...64 µs, natomiast zero 95...120 µs. Przy przerwaniu generowanym co 10 µs daje to dla jedynki 5...6 zliczeń, a dla zera 8...9 zliczeń. Różnica pomiędzy 6 a 8 jest niewielka, zwłaszcza gdy weźmie się pod uwagę, że nie zawsze przerwanie było obsłużone w 10 µs. Wypróbowałem więc inne rozwiązanie.

Sygnał na wejściu PB2 mikrokontrolera wywołuje przerwanie. W konsekwencji przerwania wywoływane są nie częściej niż co 50 µs. Biorąc pod uwagę, że przerwanie od zbocza opadającego tylko zeruje timer i uruchamia przetwarzanie A/C, to takie rozwiązanie daje w praktyce 8 razy więcej czasu na realizację programu głównego. Timer zlicza co 1 µs. Dla „1" daje to wartość timera odczytaną w przerwaniu w zakresie 52...64, dla zera 95...120. Umożliwia to pewne odróżnienie „0" od „1".

W programie można znaleźć następującą sekwencję rozkazów:

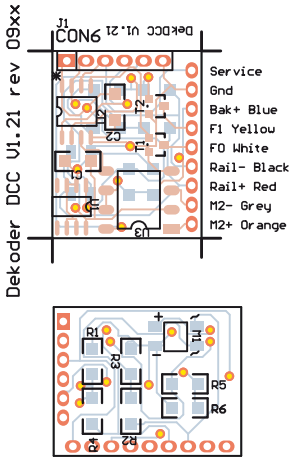
```
Speed=(((CV5 MaxV-CV2
MinV)*128)/32*Tspeed)/128+CV2
MinV;
```

Jej zadaniem jest obliczenie wartości, którą należy wpisać do rejestru kontrolnego PWM, aby lokomotywa poruszała się z zadaną prędkością (Tspeed), uwzględniając minimalne (CV2 MinV) i maksymalne (CV5 MaxV) dopuszczalne napięcie na silniku. Dziwne może wydać się mnożenie, a następnie dzielenie przez 128. Pierwotnie sekwencja ta wyglądała tak: `Speed=((CV5 MaxV-CV2 MinV)/32*Tspeed)+CV2 MinV;`. Nie chciała ona jednak działać poprawnie. Powodem były liczby ułamkowe powstające w wyniku operacji: `(CV5 MaxV-CV2 MinV)/32`. Zastosowałem więc operacje zmiennoprzecinkowe: `Speed=((CV5 MaxV-CV2 MinV)/32.0*Tspeed)+CV2 MinV;`

Ale dołączone przez kompilator biblioteki zajęły prawie połowę dostępnej pamięci Flash. Dodatkowo, czas ich wykonanie był bardzo długi. Wpadłem więc na pomysł, aby zmniejszyć błąd obliczeń przez pomnożenie wyrażenia: `(CV5 MaxV-CV2 MinV)/32` przez 100, a po wykonaniu pozostałych obliczeń podzieleniu przez 100. Precyzja obliczeń wzrosła. Przypomniało mi się jednak, że kompilator z włączona optymalizacją „O2" zamienia działania typu: *a=a*8;* na operację przesunięcia w lewo o trzy bity. Zamiast mnożyć i dzielić przez 100, użyłem operacji mnożenia i dzielenia przez 128, co kompilator zamienił na operacje przesuwania. Dzięki temu wyrażenie takie zajmuje mniej pamięci Flash i wykonuje się znacznie szybciej. Można w kodzie źródłowym zastąpić: `Speed=(((CV5 MaxV-CV2 MinV)*128)/32*Tspeed)/128+CV2 MinV;` przez: `Speed=(((CV5 MaxV-CV2 MinV)<<7)/32*Tspeed)>>7+CV2 MinV;`, ale końcowy efekt kompilacji będzie podobny (jednak nie identyczny; drugie rozwiązanie będzie zajmowało mniej pamięci).

Warto przyjrzeć się umieszczonej na **list. 1** procedurze obsługi przerwania z wejścia PB2. Jest ono wywoływane zarówno opadającym jak i narastającym zboczem sygnału DCC.

W przerwaniu sprawdzany jest stan timera i na podstawie odczytanej wartości podejmowana jest decyzja czy transmitowano bit o wartości „0", czy „1". Jeśli impuls nie mieści się w zakresie, to wyjście z przerwania jest bez zerowania timera. Dzięki temu, jeśli zakłócenie było krótkie, to istnieje szansa, że kolejne przerwanie zmieści się w zadanym przedziale czasu i bit nie zostanie stracony. Po określeniu czasu trwania bitu, odczytywana jest też wartość konwersji z przetwornika A/C. Następnie zerowany jest timer i uruchamiana konwersja A/C. W programie nie ma sprawdzenia czy przetwornik skończył przetwarzanie, ponieważ czas trwania najkrótszego sygnału jest pięciokrotnie dłuższy od czasu przetwarzania przetwornika taktowanego najniższą, możliwą częstotliwością (podział przez 16). Opcjonalnie w kodzie źródłowym, można włączyć sprawdzanie gotowości przetwornika. Może to być konieczne, gdy występuje dużo za-



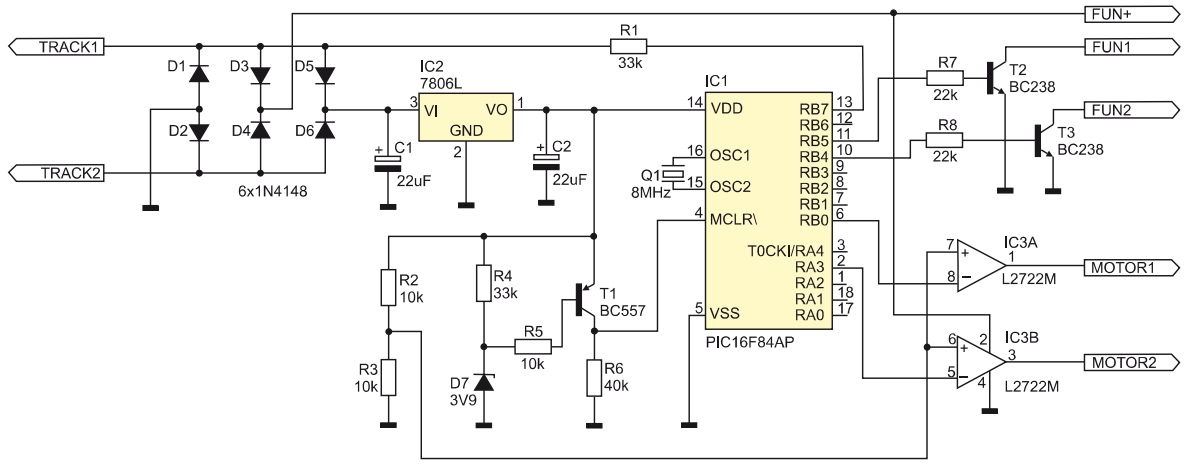
Rys. 3. Schemat montażowy dekodera

List. 1. Funkcja obsługi przerwania INT0

```
//---------------------------------------------------------------------------//
// Obsługa przerwania od wejścia INT 0
//---------------------------------------------------------------------------//
ISR (INT0 vect)
{
  static unsigned char volatile VoltR=0,VoltL=0;
  static char PozVolt=ElemVolt-1;
  unsigned char nic;
  if ( RailR == 0)                                  // Mierzymy czas poziomu (R / L - zależy od szyny)
  {
      if ( (TCNT0 > 52) && (TCNT0 < 64) )           //Odczytujemy timer
      {
        analiza danych(1);                          //Wykryto bit o wartości 1
      }
    else if ( (TCNT0 > 90) && (TCNT0 < 255) )
      {
        analiza danych(0);                          //Wykryto bit o wartości 0
        // Składamy ramkę
        }
      else if (TCNT0 > MAX LOW)
      {                                             //Gdy czas impulsu większy niż czas maks.
        TCNT0 = 0;                                  //to zerujemy timer
        return;                                     //i wychodzimy bez uruchamiania konwersji A/C
      }
    else
      {
        return;                                     //Gdy czas poza zakresem to wychodzimy z przerwania
      }                                             //nie zerując timera i nie startując AC
      VoltR = ADCL;                                 //Wynik do zmiennej
      nic = ADCH;                                   //ADLAR=0 więc dodatkowy odczyt ADCH
      ADMUX = 3;                                    //Wejście PB3 (dla V1.1 i ADLAR = 0)
      ADCSRA = (1<<ADIF) | (1<<ADEN) | (1<<ADSC) | 4; // Start konwersji
    }
  else
    {
      TCNT0 = 0;                                    //Zerujemy timer
      VoltL = ADCL;                                 //Wynik do zmiennej
      nic = ADCH;                                   //ADLAR=0 więc dodatkowy odczyt ADCH
      ADMUX =  1;                                   //Mux na wejście PB2
      ADCSRA = (1<<ADIF) | (1<<ADEN) | (1<<ADSC) | 4; //Start konwersji
      //--- Obliczenie różnicy napięć na szynach ---
      VoltTab[ PozVolt-- ] = VoltR - VoltL;         // Obliczenie różnicy napięć
      if ( PozVolt == -1 )                          //Gdy tablica pełna to średnia (flaga dla programu głównego)
      {
        PozVolt = ElemVolt-1;                       //Wskaźnik końca tablicy
        FL TabVolCompil = 1;                        //Informacja o zapełnieniu tablicy
      }
  }
}
```

Tab. 1. Funkcje rejestrów

Adres rejestru CV	Zakres wartości	Wartość domyślna	funkcja
1	0...127	3	Adres dekodera
2	0...127	30	Minimalne napięcie na silniku (0...50%)
3	0...31	10	Czas rozpędzania lokomotywy. T[ms]=CV3*8 ms*256/8 T[ms]=CV3*256 CV3=15 daje czas rozpędzania: 15*256=3,8 sekundy Wpisanie 0 wyłącza funkcję
4	0...15	3	Czas hamowania (reguły jak dla rozpędzania)
5	128...255	255	Maksymalne napięcie na silniku (50...100%) dla prędkości 31
7	0...255	17	ID wersja dekodera (tylko do odczytu, procedura odczytu jeszcze nie obsługiwana)
8	0...255	13	Przy odczycie ID producenta (procedura odczytu jeszcze nie obsługiwana) zapisanie 8 przywraca ustawienia fabryczne (tablica prędkości użytkownika nie jest modyfikowana)
9	0...3	1	Częstotliwość PWM'a sterującego silnik 0...250 Hz 1...500 Hz 2...1 kHz 3...2 kHz
11	0...255	16	Czas w 8ms od braku transmisji do wyłączenia silnika lub przejścia na jazdę analogową. To czy silnik się zatrzyma, czy będzie to jazda analogowa zależy od CV29 CV11=127 da czas 0,008s *127=1 sekunda Standardowe ustawienie daje czas 0,008*16=128 ms
27	0...7	3	Decoder Automatic Stopping Configuration (+1) Bit0=„1" Zatrzymanie gdy odchyłka dodatnia i jedzie do przodu (+2) Bit1=„1" Zatrzymanie gdy odchyłka ujemna i jedzie do tyłu (+4) Bit2=„1" zmienia reakcję na kierunek
29	0...15	2	Konfiguracja dekodera (+1) Bit0=„0" DIR normal, „1" DIR reversed (zamienione kierunki jazdy) (+2) Bit1=„0" 14 kroków, „1" 28 kroków (+4) Bit2=„0" tylko DCC (brak transmisji to stop), „1" – Analog możliwy (gdy brak transmisji) (+16) Bit4=„0" prędkość kontrolowana przez CV2,5,6, „1" indywidualna tablica prędkości
134	0...31	6	poziom napięcia po którym nastąpi zatrzymanie lokomotywy przy asymetrii na szynach.
136	0...255	80	prędkość do której zwolni lokomotywa, jeśli ustawione zwalnianie zamiast zatrzymania po wykryciu asymetrii



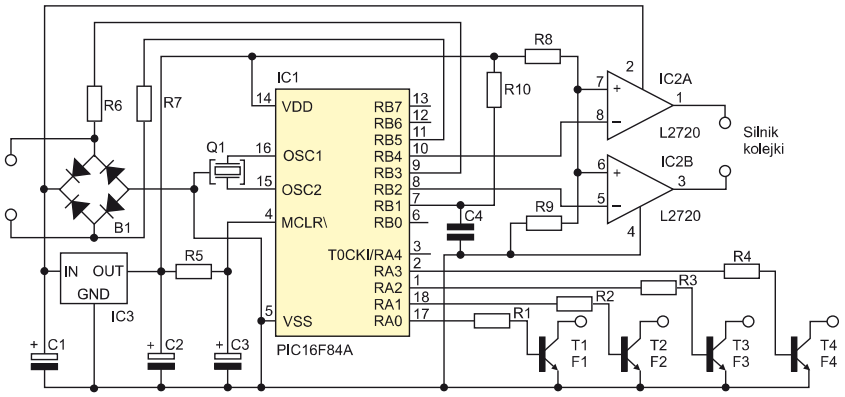
Rys. 4. Przykładowe rozwiązanie dekodera DCC dostępne w internecie

kłóceń. Wynik przetwarzania jest zapamiętywany w tablicy. Dzięki temu w programie głównym można uśrednić wyniki kilkunastu pomiarów.

Montaż

Schemat montażowy dekodera umieszczono na **rys. 3**. Montaż dekodera rozpoczynamy od strony spodniej strony płytki, lecz nie montujemy mostka. Następnie montujemy górę dekodera bez procesora. W kolejnym kroku montujemy mostek. Na koniec montujemy złącze J1 (jeśli przewidujemy programowanie procesora w działającym urządzeniu). Przed wlutowaniem procesora sprawdzamy napięcie na wyjściu stabilizatora. U3 lutujemy skracając mu wcześniej doprowadzenia. Procesor można zaprogramować przed wlutowaniem w płytkę lub w już działającym urządzeniu, korzystając ze złącza J1. Programując procesor należy odpowiedni ustawić bity konfiguracyjne:

```
WdgTimerAlwasOn = YES
BODLEVEL = 2.7V
```



Rys. 5. Przykładowe rozwiązanie dekodera DCC dostępne w internecie

```
DIV CK 8 = NO
OSC = Int RC osc 8MHz 6Ck/14Ck + 0ms
```

Należy także pamiętać o zaprogramowaniu pamięci eeprom.

Dekoder można uruchomić na stanowisku testowym lub w lokomotywie. Konfigurowanie dekodera rozpoczynamy od ustawienia w trybie serwisowym rejestru CV1. Następnie konfigurujemy CV2, CV5, CV29. Funkcje spełniane przez poszczególne rejestry opisane są w **tab. 1**.

Autor jest w trakcie oprogramowywania dekodera na wykonanego w oparciu o Atmega8. W stosunku do opisanego dekodera, ma on cztery wyjścia funkcyjne i generator dźwięków (pamięć na próbki 32 MB). Jeśli taki dekoder interesuje Czytelników, to prosimy o listy.

Sławomir Skrzyński, EP
slawomir.skrzynski@ep.com.pl