kit 2965 AVT

Zasilacz do kolejki elektrycznej – namiastka DCC

Zawsze w swojej kolejce chciałem zastosować DCC (Digital Control Comand). Jednak kupno takiego systemu lub nawet samodzielna jego budowa nie jest dla mnie ekonomicznie uzasadniona. Mam tylko dwie kolejki na makiecie. W końcu, kiedy poznałem swoje możliwości w pisaniu prostych programów, stwierdziłem, że zaprojektowanie i samodzielne wykonanie choćby prostej hybrydy zasilacza i dekodera DCC stało się realne. Budowa prezentowanego przeze mnie zasilacza nie wykracza poza podstawowe umiejętności nawet raczkującego elektronika. Wystarczy, że umie czytać schemat i trzymać prawidłowo lutownicę w ręce. Jak się okazuje w praktyce, zaprogramowanie mikroprocesora także jest bardzo proste.

W Internecie spotkałem niewiele podobnych układów i opisów, na których mógłbym się wzorować. Wszystkie były oparte na analogowych generatorach PWM. Starałem się, czytając różne fora tematyczne, wśród bełkotu i zarozumialstwa znaleźć maksymalnie wiele wskazówek. Zastanawiałem się nad sterowaniem analogowym, ale uważam, że ta droga byłaby trochę uciążliwa, więc zastosowałem procesor ATmega8. Program napisałem w poczciwym Bascomie, aby udowodnić przy okazji, że jest to doskonałe narzędzie zarówno dla tych, którzy zaczynają swoją przygodę z elektroniką, jak i dla tych bardziej „dojrzałych". Do użycia właśnie tego środowiska programowego dopingował mnie jeden przeczytany gdzieś komentarz: „że programy powinniśmy tworzyć w środowisku, które najlepiej czujemy".

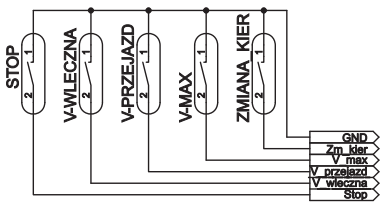
Opis układu

Jak widać na **rysunku 1**, jest to typowa aplikacja – procesor ze stabilizatorem i kwarcem. Tranzystor MOSFET T2 do regulacji napięcia i przekaźnik REL1 do zmiany biegunowości napięcia na wyjściu. Diodę D6 dodałem asekuracyjnie, można jej nie lutować, gdyż MOSFET-y mają diodę w swojej strukturze. Do zmiany kierunku jazdy nie użyłem mostka H, tylko dwuobowodowego przełącznego przekaźnika (stary poczciwy MT12, ale w prezentowanym modelu dla estetyki wstawiłem RM82). Do regulacji prędkości służy jeden MOSFET z kanałem N, wydłubany z uszkodzonej żarówki energooszczędnej. Doskonałym źródłem, by pozyskać takie tranzystory, są także przetwornice i zasilacze impulsowe, choć nie wszystkie.

Kolejnym elementem jest typowy wyświetlacz LCD 16x2, no i kilka przycisków. Do testów i do symulacji zdarzeń z makiety (**rysunek 2**) użyłem starych isostatów. Natomiast docelowo w makiecie powinny pracować kontaktrony. Superźródłem do pozyskania miniaturowych kontaktronów są stare, masywne klawiatury używane w pierwszych komputerach i starych kalkulatorach. Niewielkie rozmiary rurek tych elementów nadają się idealne do zamaskowania w torowiskach. Potencjometr, stabilizator, kondensatory i mostek prostowniczy pochodzą z jakiegoś starego, uszkodzonego monitora. Koszt procka, kwarcu i wyświetlacza to około 25zł. Obudowa, jaka jest pod ręką. Kiedyś używałem nawet mydelniczek.

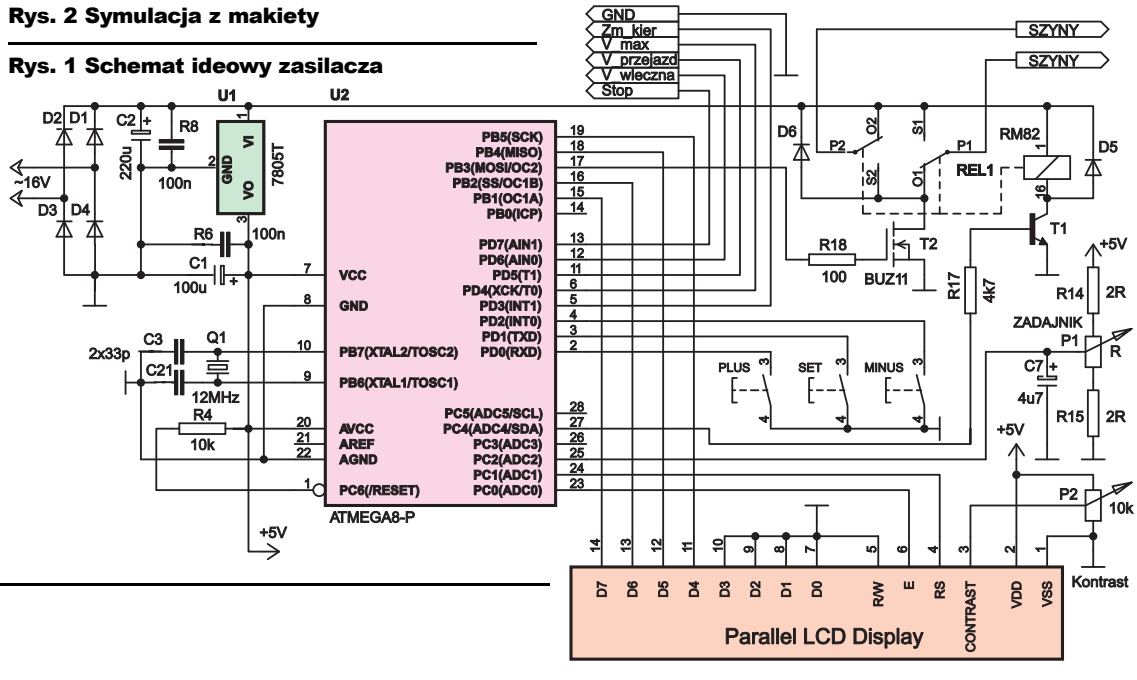
Układ możemy zasilić z czegoś co mamy pod ręką i da nam około 12...16V, a pozwoli się obciążyć do 1,5–2A. Może to być transformator czy jakiś inny zasilacz ze stałym napięciem, ponieważ wbudowany w moim zasilaczu prostownik pozwala na zasilanie byle czym bez konieczności kontrolowania biegunowości.

Zmiana napięcia wyjściowego w moim modelu opiera się na generatorze PWM i tranzystorze MOSFET T2. Aby to uruchomić, zrobiłem „doktorat". Wszystko jest proste do momentu, dopóki stosuje się typowe aplikacje i gotowe rozwiązania. Ja ambitnie chciałem płynnie regulować częstotliwość pracy mojego generatora. Niestety, nie udało mi się. Może w kolejnych wersjach. W tym przypadku jest ona stała i wynosi około 23kHz przy prescalerze ustawionym na 1. Tylko dlatego, że przy ustawionym na 8, częstotliwość pracy PWM wyniosła około 3kHz,



Rys. 2 Symulacja z makiety

Rys. 1 Schemat ideowy zasilacza



a wtedy słyszałem cichy gwizd w silnikach przy ruszaniu. Po dłuższej zabawie było to bardzo męczące. Nie umiałem zaprogramować regulowanego generatora, nawet wprowadzając timery w tryb CTC, bo z chwilą dopisania obsługi przerwań programowych, przy prescalerze ustawionym na 1 lub 8, procesor nie dawał sobie z tym rady, a zwalniając, robił wrażenie uszkodzonego. Jego reakcja wymuszała ustawienia prescalera na 256, a tym samym nici z częstotliwością powyżej 700Hz, no i jej regulacji. Więc zniechęcony i zrezygnowany ustawiłem w końcu Timer2 na stałe, jako typowy generator PWM z prescalerem 1. Ponieważ jego przerwania są sprzętowe, a nie programowe, pracuje w takich warunkach bez większych problemów. Może ktoś bardziej doświadczony podzieli się swoimi uwagami i osiągnięciami w tym zakresie. Jako 8-bitowy generator wymaga do regulacji wypełnienia w zakresie od 0% do 100%, zmiennej COMPARE (Ocr2) typu BYTE, której wartość musi się zmieniać 0–255. W moim programie operuję na zmiennych, które nie są większe niż 100 jako 100%, dlatego musiałem wprowadzić pewne przeliczenie widoczne na **listingu 1.** To podaję jako rozwiązanie dla tych, którzy szukają oszczędności w kodzie podczas pisania „opasłych" programów. BASCOM nie chciał mi przemnożyć zmiennej BYTE przez ułamek, wymuszając deklarację zmiennej jako zmiennoprzecinkowej, która pochłaniała ogromną ilość pamięci, więc zastosowałem inną metodę. Zrobiłem przeliczenia na liczbach całkowitych, uzyskując zamierzony wynik jako iloczyn z ułamkiem. W tym wypadku zmienna była typu WORD i zaoszczędzonych kilka bajtów pamięci.

```
'nie chce tego obliczyc mnozy tylko przez 2
'V_na_szynach = V_do_obliczen * 2,55

'ale tak już nie protestuje
   V_na_szynach = V_do_obliczen * 255
   V_na_szynach = V_na_szynach / 100

Ocr2 = V_na_szynach
```

Listing 1

Do zadawania prędkości są dwa dostępne sposoby. Jeden z nich jest możliwy dzięki zadajnikowi opartemu na potencjometrze, a drugi automatyczny, realizowany z sygnałów pochodzących z makiety, które równie dobrze możemy dobudować w formie przycisków w obudowie, oprócz tych, które musimy wstawić aby poruszać się po programie. Ale po kolei.

Do regulacji prędkości pierwszym sposobem wykorzystałem wbudowany w procesorze przetwornik analogowo-cyfrowy (A/D), który dokonuje pomiarów napięcia na suwaku potencjometru. W napisanym przez siebie programie opierałem się o zmienne typu BYTE, których wartości nie osiągały więcej niż 100. Jak już wcześnie wspomniałem, jako odniesienie jest poziom 100%. W ten sposób łatwo dopasowuje się różne wielkości fizyczne. Potencjometr mógłbym zasilić bezpośrednio napięciem 5V, ale wartości odczytane z dziesięciobitowego przetwornika A/D osiągałyby wartość 0...1023 i trzeba by je przeliczać, dzieląc wynik przez jakąś stałą, co oznacza kilka dodatkowych linijek kodu. Ja zrobiłem analogowy dzielnik napięcia, abym nie musiał tego przeliczać. Dzięki temu wynik pomiaru to gotowa wartość na potrzeby programu. Ponieważ założyłem, że środek położenia suwaka to będzie moje zero, a skrajne położenia będą odpowiadały 100% prędkości w obu kierunkach, więc wartość, jaką musiałem uzyskać z pomiaru przez przetwornik A/D przy skrajnych położeniach pokrętła, to około 200 jednostek pomiarowych (po 100 dla każdego kierunku). Zgodnie z założeniami na dolutowanych opornikach powinna się odkładać wartość około 800 „jednostek", czyli po 400 dla każdego. Tak więc przez analogię i proporcję dobrałem oporniki: jeśli np. mamy pod ręką potencjometr o wartości 10kΩ, to oporniki w dzielniku powinny mieć wartość po 20kΩ każdy. To są rozważania teoretyczne, praktyka mówi, że trzeba wlutować dwa opory po 18kΩ. Jeśli natomiast mamy potencjometr 47kΩ, to teoretycznie oporniki powinny mieć wartość po 94kΩ, a praktycznie 86kΩ.

We fragmencie **listingu 2** widać, jak zrealizowałem zadajnik prędkości. Aby program wykrywał kierunek zadania, dokonałem podziału zmierzonego napięcia na strefy. Jak już wspomniałem, pełny zakres pomiarowy przy 10-bitowym przetwarzaniu daje wynik 0...1023, więc przy podziale na połowę daje to 500 z groszami. Ponieważ w rzeczywistości bardzo trudno byłoby ustawić suwak dokładnie w położeniu środkowym, wprowadziłem programowo martwą strefę. W moim programie wynosi 20. Czyli wartości od 490 do 510 układ odczytuje jako prędkość równą zeru. Dopiero przekręcenie suwaka w potencjometrze tak, by wartość mierzona była większa od 510 lub mniejsza od 490, wpływa na regulację prędkość. Jak się okazało przy testach, ta martwa strefa wystarcza w zupełności. Dodatkowo zmiennymi bitowymi określiłem kierunek zadania, bo od tego miejsca potencjometr będzie zadawał prędkość w przód i w tył, kiedy przekręcimy gałką w prawo lub w lewo od jego środka. O ile z zadaniem prędkości do przodu, czyli wartościami mierzonymi od 510 w górę, nie ma większego problemu, bo można je wprost

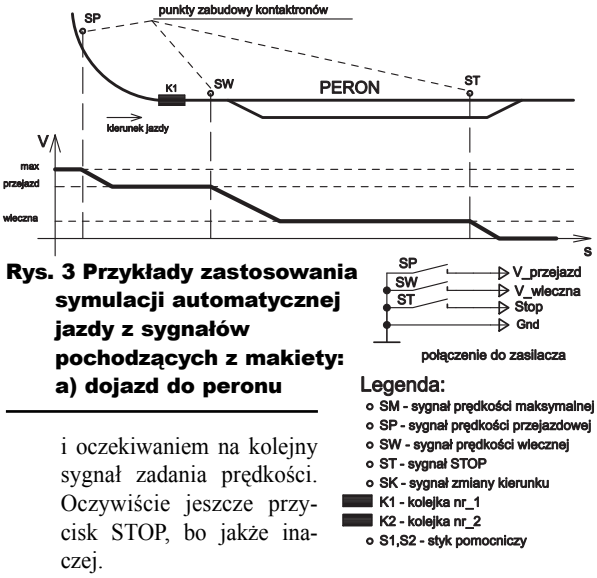
```
Sub Pomiar
Joy = Getadc(2)
If Joy >= 490 And Joy <= 510 Then 'martwa strefa
       Kier_przod = 0
       Kier_tyl = 0
       Joy = 0
       Else
       If Joy > 510 Then
          Joy = Joy - 510
          Kier_przod = 1
          Kier_tyl = 0
          Else
          If Joy < 490then
             Joy = 490 - Joy
             Kier_tyl = 1
             Kier_przod = 0
          End If
       End If
End If
   If Joy > 100 Then
      Joy = 100
   End If
Return
End Sub
```

Listing 2

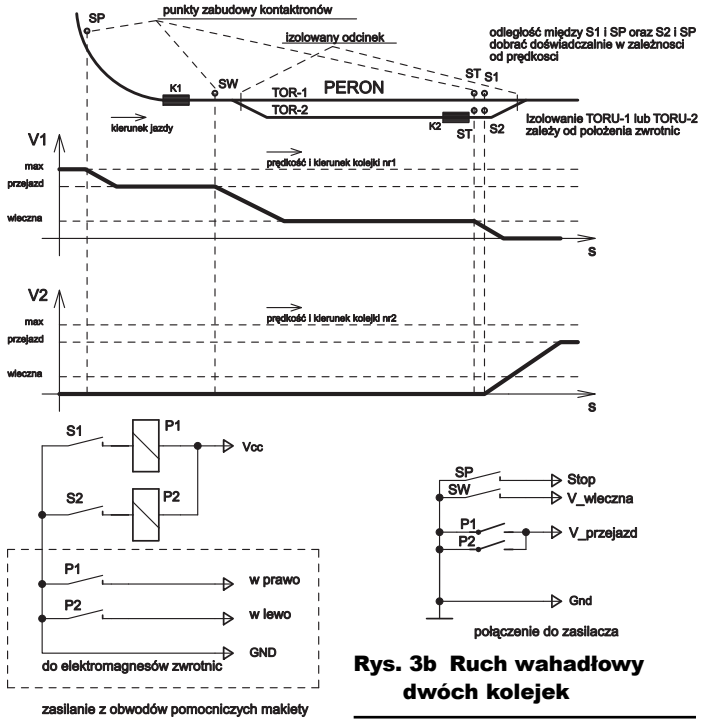
interpretować i wprowadzać do programu, to zadanie prędkości w tył zgodnie z tą ideą będzie niewskazane, gdyż będą się one zaczynały od 490 i malały w dół. Kolejka zaczęłaby jazdę od większej prędkości do mniejszej. Tak więc „odwróciłem" ten pomiar, odejmując od 500 zmierzoną wartość z A/D, uzyskując w ten sposób odpowiedni przyrost zadania prędkości w drugą stronę. Dzięki tym zabiegom uzyskałem zamierzony diagram mojego zadajnika w przód i w tył.

Drugim sposobem zadawania prędkości jest jazda automatyczna, czyli skokowa, zależna od sygnałów pochodzących z makiety wymuszanych za pomocą wbudowanych np.: kontaktronów lub innych styków, usytuowanych w torowisku. Ale może ona być realizowana również z dodatkowych przycisków dobudowanych w pulpicie. Aby zrealizować ten sposób jazdy, przyjąłem kilka prędkości jako punkt wyjścia dla tego rozwiązania. Pierwszą z nich jest prędkość maksymalna i dotyczy ona także sterowania ręcznego. Z chwilą wymuszenia tej prędkości przyciskiem lub kontaktronem z makiety, kolejka będzie się rozpędzała do tej właśnie wartości, zgodnie z zaprogramowanym przyśpieszeniem, na które również możemy wpłynąć. O tym później. Kolejną zmienną jest prędkość przejazdowa. Może ona być równa prędkości maksymalnej, ale większa od niej nie będzie, bo ogranicza to program. Ta prędkość wprowadza większe urozmaicenie i przełamuje monotonię podczas jazdy kolejki po makiecie.

Następnie pomyślałem o dojazdach do różnych punktów przystankowych, w których kolejka musi się zatrzymać i zastosowałem prędkość, którą nazwałem „wleczną". Musi się to odbyć tak, aby maksymalnie przybliżyć realność poruszania się składu, a jednocześnie ograniczyć karkołomne przeliczania odległości od punktu zwalniania do punktu zatrzymania i ciągłemu dobieraniu miejsca zabudowy kontaktronów na makiecie. Dopisałem jeszcze procedurę dla zmiany kierunku jazdy. Z chwilą jego wymuszenia nastąpi zmiana kierunku jazdy z jednoczesnym zwalnianiem do prędkości zero



Rys. 3 Przykłady zastosowania symulacji automatycznej jazdy z sygnałów pochodzących z makiety: a) dojazd do peronu



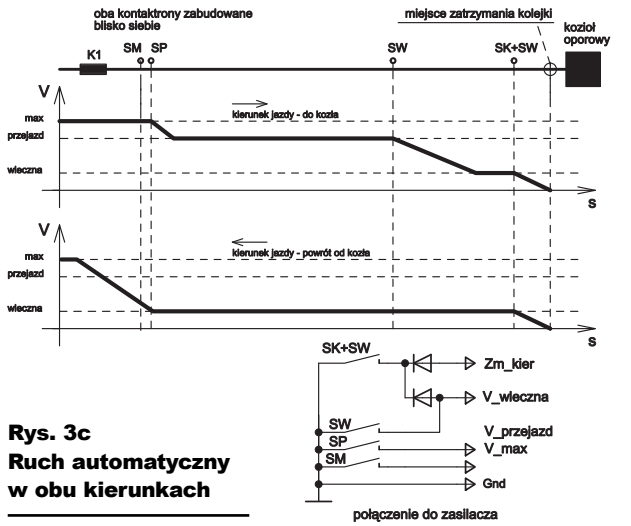
Rys. 3b Ruch wahadłowy dwóch kolejek

i oczekiwaniem na kolejny sygnał zadania prędkości. Oczywiście jeszcze przycisk STOP, bo jakże inaczej.

Na **rysunkach 3a,b,c** podałem kilka przykładów zastosowania symulacji automatycznej jazdy z sygnałów pochodzących z makiety. Dodam, że mój zasilacz zasila tylko kolejki. Do sterowania zwrotnicami i innym osprzętem jest wymagany inny, dodatkowy zasilacz do tego przeznaczony. Do jednoczesnego podawania kilku sygnałów z jednego punktu, czyli do kolejki i zwrotnic jednocześnie, należy zastosować przekaźniki jako galwaniczną izolację, aby nie robić jakichkolwiek zakłóceń lub zwarć. W przykładach oprócz schematycznego torowiska i przykładów zabudowania czujników – kontaktronów, zamieściłem wykresy prędkości w funkcji drogi dla lepszego zobrazowania poruszającej się kolejki. W pierwszym przypadku pokazałem dojazd do peronu jako jedno z najprostszych zastosowań. Zasilacz dostaje kolejno komendy zwalniania i kolejka wjeżdża na peron z minimalną prędkością, którą nazwałem „wleczną", aby po komendzie STOP, zatrzymać się na bardzo krótkim odcinku. W następnym przykładzie pokazałem ruch wahadłowy dwóch kolejek, wykorzystując dwa tory przy peronie jako izolowane odcinki, zwierane lub rozwierane przez zwrotnice (takie posiadam na makiecie). Jedna kolejka „oczekuje" na wjazd pierwszej, ponieważ nie dostaje napięcia. Ta będąca w ruchu zatrzymuje się w wyznaczonym punkcie i w ostatnim momencie zmienia położenie zwrotnic izolując swoje torowisko, jednocześnie wydając do zasilacza komendę przyśpieszenia. Lecz to polecenie wykonuje już ta druga, jako że jej tor jest pod napięciem. W następnym przykładzie pokazałem możliwość jazdy tam i z powrotem. Kolejka realizuje zwalnianie, ponieważ zbliża się do kozła oporowego. Przed zatrzymaniem zasilacz dostaje komendę zmiany kierunku jazdy z jednoczesnym zadaniem prędkości „wlecznej", czyli po zatrzymaniu w wyznaczonym miejscu odczeka kilka sekund i będzie realizowała jazdę z powrotem. Tu pokazałem kombinowane połączenia komend do zasilacza poprzez zwykłe diody sterownicze. Ten przykład pokazuje, że kolejka może poruszać się po torze zamkniętym, nie musi to być wcale pętla. Mam nadzieję, że te przykłady zainspirują przyszłych modelarzy.

Jak będzie się rozwijała „edukacja" mojego zasilacza, zależy już tylko od opinii moich kolegów, którzy testują mój układ lub od Czytelników zainteresowanych tym projektem. Tyle na temat zadawania prędkości.

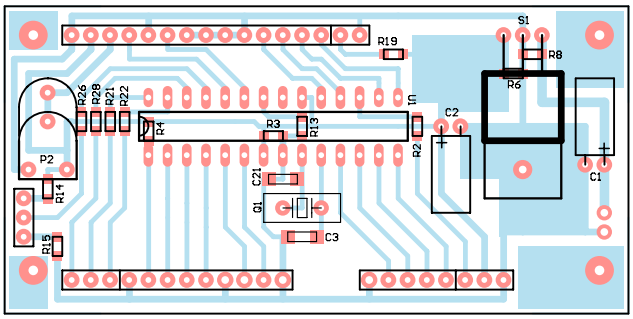
Kolejnym najważniejszym podprogramem zasilacza jest algorytm jazdy. W moim programie jest on liniowy, bo tak było najprościej, ale mam kilka pomysłów, aby „zakrzywić" trochę charakterystykę, nadając większy realizm poruszania się kolejek na makiecie. W założeniach do diagramu jazdy przyjąłem takie zmienne: *przyśpieszenie, opóźnienie* jako różne wartości, *krokowy przyrost prędkości, prędkość startowa, prędkość maksymalna*. Zastosowałem jeszcze wymuszony czas postoju przy prędkości zerowej podczas zmiany kierunku jazdy, około trzech sekund tak, aby kolejka nie ruszała od razu w przeciwnym kierunku po zatrzymaniu. Zmiana tego czasu nie jest dostępna z menu zasilacza ze względu na brak miejsca w pamięci procesora i w proponowanej wersji wynosi około 3 sekund.



Rys. 3c Ruch automatyczny w obu kierunkach

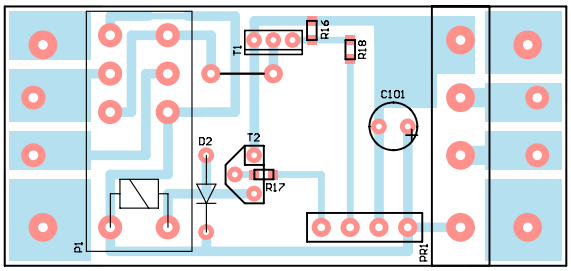
Prędkość startowa jest wprowadzona tylko po to, aby zaczynając jazdę od zera, silnik dostał impuls do startu. Jak mówi doświadczenie, silniki nie rozpoczną płynnego wirowania przy regulacji napięcia od zera ze względu na swoją bezwładność. Dlatego stosuje się wyższe napięcie startowe, aby nadać większą płynność poruszania. Po obserwacjach pracy mojego zasilacza i poruszania się kolejek na makiecie nie byłem do końca usatysfakcjonowany. Ta opcja w moim projekcie jest jeszcze w trakcie dopracowywania, bo chodzą mi po głowie różne pomysły. W moim programie wartość startowa jest tylko dodawana jako jednostkowa początkowa wartość skokowa ale tylko do wartości zerowej, czyli kiedy kolejka stoi. Później ruch odbywa się zgodnie z zadaniem prędkości i z zadanym krokiem. Podczas zwalniania nie ma skoku napięcia podobnie jak przy ruszaniu. Zmiana tej wartości jest dostępna w MENU. Zastanawiałem się też nad takim sposobem, aby na krótką chwilę, ułamek sekundy, pobudzić silnik większym napięciem, a potem od ustalonej wartości minimalnej rozpocząć rozpędzanie. Na razie ten pomysł jest na tapecie.

Prędkość krokowa to zmienna, o którą będzie pomniejszana lub powiększana prędkość jazdy podczas przyśpieszania bądź zwalniania pociągu. W początkowej fazie pisania programu uzależniłem ją od liczby kroków i prędkości maksymalnej. Wynik tej operacji był często ułamkiem, zaokrąglanym przez procesor, więc rozsądnym rozwiązaniem okazało się zastosowanie przyrostu prędkości jako zmienną wprowadzaną ręcznie przez użytkownika w zakresie od 1 do 20. Przy prędkości maksymalnej równej 100 da to maksymalnie 100 kroków lub minimalnie



Rys. 4 Schemat montażowy

Rys. 5 Schemat montażowy



5 kroków. To było najlepsze rozwiązanie i potwierdziła to praktyka.

Opóźnienie i przyśpieszenie są osobnymi zmiennymi. Każdą z nich możemy w dowolnej chwili zaprogramować. Wskazaną w MENU ustawień wartość należy przemnożyć przez 0,1 sekundy, a tak otrzymany wynik będzie odstępem czasu w wykonaniu kolejnych kroków. Do realizacji tej opcji użyłem jednego z dostępnych timerów. Nie korzystałem nigdzie w programie z funkcji WAIT, czy WAITMS, aby nie spowolnić pracy programu, tylko przyciski są obsługiwane z takim opóźnieniem, ponieważ w założeniach mają być dwa jednakowe kanały obsługujące niezależnie dwie kolejki. Zastosowanie ATmegi16 pozwoli na taką pracę doskonale. Wspomniany Timer pracuje w kółko bez przerwy z wymuszonym przerwaniem co 0,1s. Do odmierzenia odpowiedniego odcinka czasu jest wykorzystana odpowiednia flaga i zmienna pomocnicza „pobudzana" tylko w momencie realizacji przyśpieszania lub zwalniania. Z każdą chwilą jej programowej aktywności jest zwiększana o 1 w momencie przerwania timera i porównywana z wartością wpisaną do pamięci procesora przez użytkownika. Maksymalna wartość, którą możemy wprowadzić, wynosi 20. Uważam, że jest to o wiele za dużo, ale wprowadzenie tak szerokiego marginesu wpisywania zmiennych zostawiłem specjalnie. Może ktoś będzie potrzebował takiego zasilacza do czegoś innego niż makieta.

Przykład 1: przyjmując, że prędkość krokowa będzie równa 1, a prędkość maksymalna równa 100 i wprowadzone przyśpieszenie przyjmie wartość 20, to czas potrzebny na rozpędzenie kolejki do pełnej prędkości wyniesie: (20x0,1)x(100/1)=200 sekund, to znaczy ponad 3 minuty.

Przykład 2: prędkość maksymalna 100, prędkość krokowa 5, 100/5=20, to daje 20 kroków z przyśpieszeniem 1, więc czas pełnego rozpędzania: (1x0.1)x(100/5)=2 sekundy.

Naprawdę jest to fantastyczna zabawa. Dzięki takim modyfikacjom możemy wprowadzić dane jako dwie różne charakterystyki, i przełączając je między sobą w MENU głównym, zasymulujemy jazdę kolejki bez wagonów, czyli szybko reagującej na zmiany zadania prędkości lub ruch całego „załadowanego" składu, który bardzo powoli będzie się rozpędzał i oczywiście bardzo wolno zwalniał. Każda taka charakterystyka może mieć swój numer widoczny przy pozycji LOK:x.

Na poruszanie się po MENU pozwalają nam przyciski: PLUS, MINUS i SET. Dzięki nim możemy wyświetlić kolejno wszystkie zapisane zmienne, a wciskając SET, zmienić wartość tej, która jest w tym momencie wyświetlana. Każda posiadana przez nas kolejka może mieć przypisane indywidualne zmienne, które możemy szybko przełączać, wybierając odpowiedni jej numer, bo są one przechowywane w wewnętrznej pamięci EEPROM. Każdy taki pakiet, w skład którego wchodzą: prędkości: maksymalna, przejazdowa, „wleczna", krokowa, prędkość startowa, oraz przyśpieszenie i opóźnienie, jest reprezentowany na wyświetlaczu w postaci kolejnej cyfry przy napisie LOK:x widocznej w jednej z pozycji MENU oraz informacyjnie na stałe przy literze J:x (joystick), która pokazuje, że zadajemy prędkość z potencjometru, lub literze M:x (makieta), symbolizując pracę w automacie. Aby kompleksowo wpisywać zmienne dla kolejnych charakterystyk, musimy wejść poprzez MENU SET i dokonać wszystkich ustawień po kolei, o które program zapyta. Tylko w tym trybie możemy dodać kolejną charakterystykę zapisaną jako kolejna lokomotywa (LOK). Na wyświetlaczu pod hasłem MEM:x pojawi się wtedy kolejna liczba i będziemy wiedzieli, ile mamy wpisanych łącznie charakterystyk. Maksymalnie możemy ich wpisać 15.

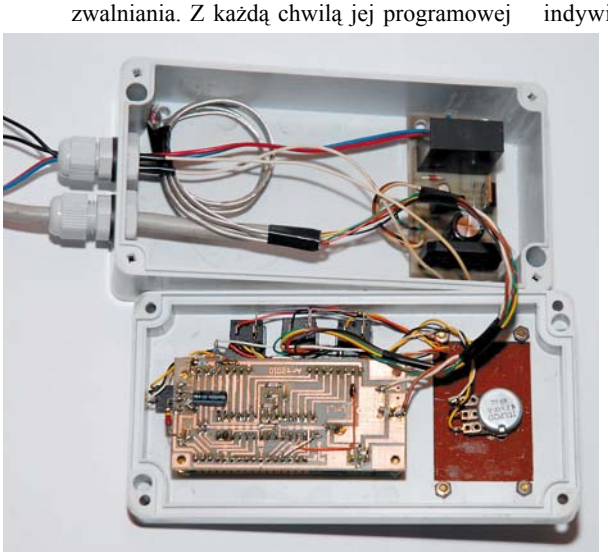
Montaż i uruchomienie

Na **rysunkach 4** i **5** przedstawione są rysunki montażowe. Montaż jest klasyczny i nie wymaga szerszego omawiania.

Jak już wspomniałem, program został napisany w Bascomie (można go ściągnąć z Elportalu) i uruchomiony prawie całkowicie w jego symulatorze. Do zaprogramowania procesora użyłem programatora ISP podpinanego do złącza LPT komputera. Na temat ustawiania fusebitów i tym podobnych nie będę się rozpisywał, bo to inny temat, poruszany wielokrotnie na łamach EdW. Podczas pierwszego programowania procesora należy wykasować pamięć EEPROM. Jest to potrzebne podczas pierwszego uruchomienia zasilacza. Dzięki temu program zorganizuje sam sobie pamięć i ją odpowiednio uzupełni. Po podaniu zasilania, układ przeniesie nas automatycznie do MENU SET – ustawień, gdyż tego wymaga do poprawnej pracy. Wtedy też musimy oczywiście zapisać choćby jedną charakterystykę dla naszej kolejki. Możemy to zrobić na oko, ponieważ w każdej chwili możemy te dane pozmieniać dzięki szybkiemu dostępowi z MENU głównego. Starałem się tak dostosować interfejs, aby był czytelny i intuicyjny. Myślę, że z tym nikt nie będzie miał problemów. W końcowej fazie wprowadzania danych program zapyta nas, czy wprowadzić dane dla następnej charakterystyki pod kolejnym zapisanym numerem. Jeśli nie, to zapisze wpisane wartości i wskoczy do MENU głównego. To pozwoli na tak długo oczekiwaną zabawę. Będę czekał na opinie z propozycjami kolejnych zmian i udogodnień lub też niedociągnięć.

Mam już kolejny pomysł, aby taki zasilacz jako moduł wstawić do zdalnie sterowanego samochodu, nadając mu ciekawy realizm w poruszaniu.

Artur Bizoń
artek.bi@interia.pl



Wykaz elementów

R18,R19	100Ω
R17	4,7kΩ
R14,R15	patrz tekst
R4,R16	10kΩ
R2,R13,R21,R22,R26,R28	0Ω
P1	patrz tekst
P2	10kΩ
T1	BUZ11, IRF540
T2	NPN, np. BC558
D1,D3,D4,D5	mostek
D2	dioda 1A
D6	dioda szybka
C1, C101	100µF/16V
C2	220µF/35V
R3,R6,R8	100nF
C5,C21	33pF
Q1	12MHz
U1	ATmega8
S1	LM7805
REL1	12V, np. RM82
Wyświetlacz LCD 2x16	