Zabawka sterowana pilotem RC5

na procesorze AT89C2051

kit 2441 AVT

Do czego to służy?

Do niedawna odpowiedź na powyższe pytanie byłaby prosta i brzmiała: "Do zabawy!". Niestety, a właściwie na szczęście czasy się zmieniły i nieprędko doczekamy się w Elektronice dla Wszystkich opisu projektu wykorzystującego procesor 89C2051, który służyłby tylko i wyłącznie do zabawy. Naszym głównym celem, któremu podporządkowaliśmy wszystkie inne działania, jest nadrobienie wieloletnich opóźnień i zapoznanie Czytelników z tajnikami techniki mikroprocesorowej. Dlatego też program sterujący pracą naszej zabawki został bardzo szczegółowo omówiony i Studenci BASCOM College mogą traktować ten opis jako dodatkowe uzupełnienie nauki w College'u.

Proponowany układ jest bardzo prosty do wykonania i składa się zaledwie z jednego układu scalonego, oczywiście procesora AT89C2051 i garstki elementów dyskretnych. Jest to kolejny przykład prostego urządzenia spełniającego dość skomplikowane funkcje, a którego możliwości zależą od zaszytego w procesorze programu.

Nasza zabawka jest napędzana dwoma silnikami pojazdu mechanicznego o konstrukcji podobnej do opisywanych niegdyś w Elektronice dla Wszystkich" słynnych "raabowozów" może być sterowana za pomocą dowolnego pilota pracującego z kodem RC5. Poza rodzajem emitowanego kodu nie ma żadnych ograniczeń co do rodzaju pilota: może to być urządzenie współpracujące zwykle z telewizorem, magnetowidem, kamerą wideo lub też pilot wykonany samodzielnie (np. AVT-849 - uniwersalny pilot umożliwiający wysłanie dowolnego kodu do dowolnego odbiornika RC5).

Nie ma też jakichkolwiek ograniczeń co do klawiszy, za pomocą których będziemy sterować naszym pojazdem. Po prostu, nasza zabawka zaraz po uruchomieniu musi "naumieć się" kodów, jakimi będzie sterowana i dopiero po podporządkowaniu jej funkcji odpowiednim klawiszom nadaje się do dalszej zabawy.

Zabawka może wykonywać następujące poruszenia:

1. Jazda do przodu
2. Jazda do tyłu
3. Skręt w lewo
4. Skręt w prawo
5. Skręt to tyłu w lewo
6. Skręt to tyłu w prawo
7. Obrót dookoła osi w lewo
8. Obrót dookoła osi w prawo
9. Zatrzymania silników
10. Funkcja dodatkowa, włączana naprzemiennie jednym z przycisków.

Nasze pociechy bywają dość roztrzepane i pozostawianie przez nie zabawek z włączonym zasilaniem należy właściwie do reguły. Układ został skutecznie zabezpieczony przed niepotrzebnym rozładowaniem baterii lub akumulatorów. Jeżeli pojazd przez jakiś czas nie otrzymuje żadnego polecenia, to po wydaniu sygnału ostrzegawczego przechodzi w "stan spoczynku", czyli wyłącza wszystkie aktywne odbiorniki energii i wprowadza procesor w stan IDLE, to jest tryb pracy z obniżonym poborem energii. Oczywiście, pierwsza ważna komanda wydana z pilota natychmiast "budzi" zabawkę z chwilowego letargu!

Jedyną trudnością, na jaką napotkacie podczas konstruowania zabawki będzie jak zwykle wykonanie części mechanicznej. Układ prototypowy został wykonany z wykorzystaniem przerobionych uszkodzonych serwomechanizmów modelarskich, które po usunięciu części elektronicznej i mechanicznych zabezpieczeń doskonale nadają się do pracy jako układy napędowe. Jednak w praktycznym wykonaniu takie rozwiązanie byłoby ekonomicznym idiotyzmem (chyba że ktoś posiada na składzie dwa niepotrzebne serwa z uszkodzoną częścią elektroniczną). Sądzę jednak, że zdobycie odpowiednich przekładni pochodzących z popsutych, tandetnych zabawek "Made in Taiwan" nie powinno nikomu nastręczyć większych trudności. Idealnym rozwiązaniem byłoby wykonanie naszej zabawki na bazie modelu czołgu i zastosowanie napędu gąsienicowego. Zauważcie, że napęd modelu pojazdu wzorowany na napędzie gąsienicowym jest najprostszą metodą rozwiązania, a właściwie ominięcia licznych problemów konstrukcyjnych, na jakie napotkalibyśmy przy budowaniu mechanizmu skręcania kół, podobnego do stosowanego w samochodach.

Jak każdy twór rąk ludzkich, nasz układ posiada także i wady. Najważniejszą z nich jest fakt, że po wyłączeniu zasilania zabawka zapomina wszystko, czego się nauczyła i po powtórnym jej uruchomieniu musimy od nowa instruować procesor, jakim kodom pilota podporządkować ma wykonywane funkcje. Nie jest to jednak zaniedbanie konstruktora, ale świadome działanie mające na celu zmniejszenie kosztów wykonania układu. Zresztą, uczenie zabawki to także ... dobra zabawa!

Jak to działa?

Popatrzmy teraz na schematy, zapoznajmy się z prostą częścią hardware'ową urządzenia, a następnie zastanowimy się, w jaki sposób zmusić martwą jeszcze część sprzętową układu do działania. Przeanalizujmy najpierw **rysunek 2**, który przedstawia część wykonawczą układu.

Część wykonawcza naszej zabawki jest typowo skonstruowanym sterownikiem dwóch silników prądu stałego, sterowanym za pomocą czterech sygnałów cyfrowych. Silniki włączone (CON1 i CON2) są w przekątne

mostków utworzonych przez tranzystory mocy typu BD139 i BD140. Do złącza CON5 dołączony jest układ mikroprocesorowy przedstawiony na **rysunku 1**, z którego budową zapoznamy się za chwilę. Rozpatrzmy teraz, co się stanie, jeżeli na przykład na wejściu 5 CON5 pojawi się stan wysoki. Łatwo zauważyć, że konsekwencją tego faktu będzie spolaryzowanie bazy tranzystora T1, a także tranzystorów T7 i T11. Prąd elektryczny popłynie na drodze: + zasilania, tranzystor T11, uzwojenie silnika dołączonego do złącza CON1, tranzystor T7 i masa zasilania. Silnik przyłączony do CON1 zacznie obracać się umownie w stronę obrotu wskazówek zegara. Ponieważ nasz pojazd posiada dwa silniki napędowe, zacznie skręcać (umownie) w lewo. Wyślijmy teraz stan wysoki na dwa wejścia złącza CON5: 5 i 3. Włączone teraz zostaną dwa tranzystory: T1 i T4, co spowoduje także przewodzenie tranzystorów T11, T7, T10 i T8 i obracanie się dwóch silników w tę samą stronę. Nasz pojazd zacznie poruszać się do przodu (lub do tyłu, ponieważ kierunek ruchu zostanie ostatecznie ustalony doświadczalnie podczas montażu pojazdu).

Sądzę, że uważni Czytelnicy zauważyli już pewne niebezpieczeństwo, tkwiące w naszym układzie. Co bowiem się stanie, jeżeli stan wysoki pojawi się jednocześnie na wejściach 5 i 4 CON5? Ano, będzie to piękne zwarcie w układzie spowodowane jednoczesnym przewodzeniem wszystkich tranzystorów mostka! Oczywiście, przy poprawnie napisanym programie taka sytuacja nie powinna się przydarzyć, ale ... nie wszystkie programy napisane są od razu poprawnie. Musimy wziąć także pod uwagę pewną cechę mikroprocesorów '51:

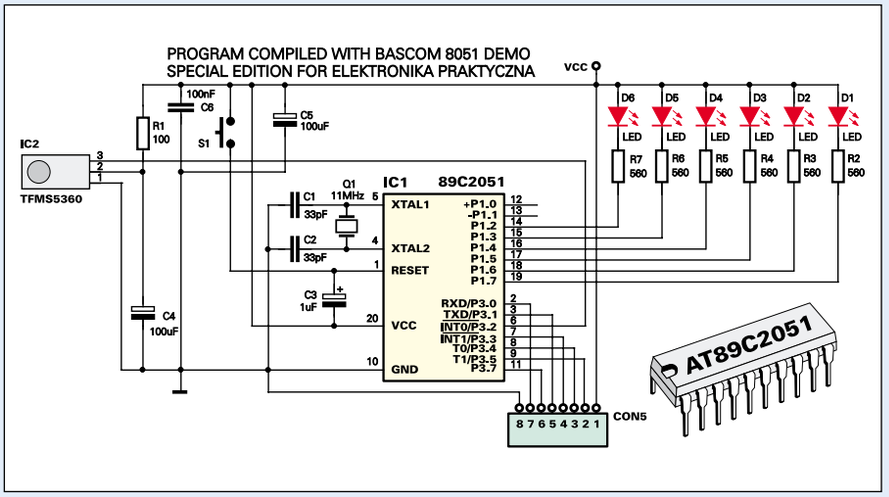
W czasie, kiedy procesor znajduje się w stanie "RESET", na wszystkich jego wyjściach pojawia się stan wysoki. Zjawisko to zachodzi zawsze w momencie startu każdego systemu mikroprocesorowego opartego na procesorze 2051!

Aby więc ustrzec się przed zwarciem i jego zwykle przykrymi konsekwencjami, do części wykonawczej naszej zabawki dobudowany został układ z bramkami NAND zawartymi w strukturze IC1 i tranzystorem T14. Zauważmy, że warunkiem zasilania silników od strony minusa zasilania jest przewodzenie tranzystora T14. Bramki IC2A i IC2B wykrywają stany zakazane, które mogłyby wystąpić na wyjściach części sterującej. Powstanie stanu niskiego na wyjściu jednej lub obu tych bramek powoduje natychmiastowe wyłączenie tranzystora T14 i wyeliminowanie niebezpieczeństwa totalnej katastrofy.

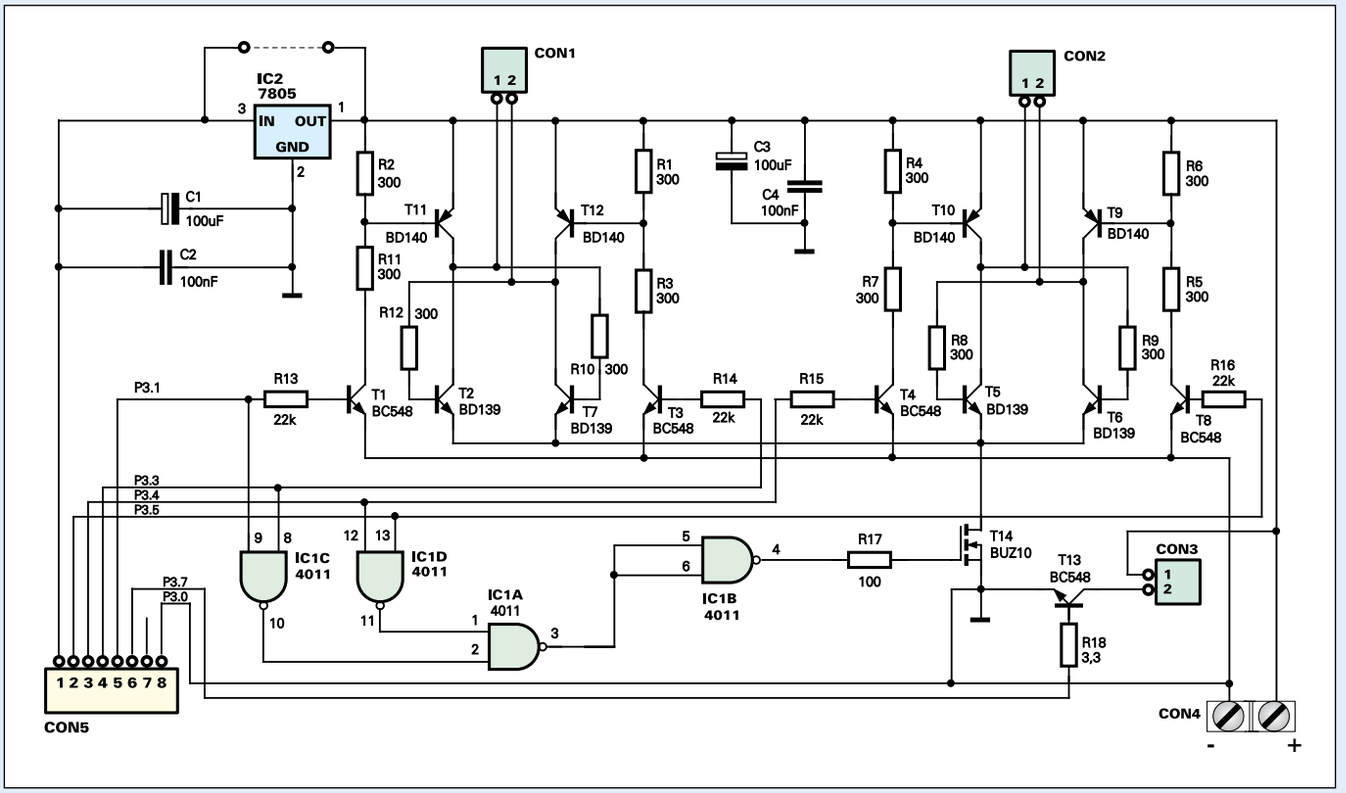
Układ może być zasilany napięciem stałym o wartości 5 ... 16VDC, uzależnionym głównie od typu zastosowanych silników. Z tego też względu stabilizator napięcia IC2 jest elementem opcjonalnym, i w przypadku korzystania z napięcia zbliżonego do 5V nie musi być stosowany.

Tranzystor T13 może włączać lub wyłączać układ dodatkowy: sygnalizator optyczny, akustyczny, jakiś silnik realizujący dodatkową funkcję, słowem wszystko, co przyjdzie Wam do głowy.

Rys. 1 Sterownik



Rys. 2 Część wykonawcza



Popatrzmy teraz na rysunek 1, na którym została pokazana część sterująca układu. Niewiele jest tu do skomentowania, aby zrozumieć zasadę działania tego fragmentu zabawki, musielibyśmy zajrzeć do wnętrza procesora, co za chwilę uczynimy. Wystarczy tylko zapamiętać, do których wyjść - wejść procesora dołączone są diody sygnalizacyjne, na których muszą pojawiać się stany logiczne powodujące określone zachowanie się pojazdu, i do jakiego wejścia dołączony jest dekoder RC5. Sądzę, że warto tylko wspomnieć o roli, jaką spełnia kondensator C3 dołączony do wejścia RESET procesora. Kondensator taki występuje w większości **prostych** systemów mikroprocesorowych, a jak do tej pory nie mieliśmy okazji wyjaśnić, do czego właściwie jest potrzebny.

Otóż każdy procesor po włączeniu zasilania musi przejść przez stan resetu, czyli wyzerowania układu, wyczyszczenia zawartości pamięci RAM i rejestrów wewnętrznych.

Wymuszenie na wejściu RESET stanu wysokiego przez okres co najmniej dwóch cykli maszynowych powoduje wstrzymanie pracy procesora i ustawienie wyjść wszystkich portów w stan wysoki. Po pojawieniu się stanu niskiego na wejściu RESET procesor podejmuje normalną pracę, wykonując od początku instrukcje zawarte w sterującym nim programie. W stanie RESET pamięć RAM procesora zostaje wyczyszczona! Uwaga: ta zasada dotyczy procesorów rodziny 'X051. Inne typy procesorów (np. AVR) zerowane są niskim poziomem logicznym.

Programowanie

Bierzmy się zatem do pisania programu obsługującego kolejne wcielenie "raabowozu". Na samym początku musimy, jak zwykle, zadeklarować wszystkie używane w programie zmienne. Zacznijmy od wartości, których nadanie przez pilota spowoduje takie, a nie inne reakcje układu. Będą to wartości z zakresu od 0 do 63, a zatem musimy zadeklarować je jako "BYTE" (maksymalna wartość 255). Takich zmiennych będziemy używać 10: 9 dla określenia sposobu poruszania się pojazdu i jedną, odpowiedzialną za włączanie i wyłączanie funkcji dodatkowej. Po deklaracji zmiennych określimy konfigurację sprzętową układu, zadecydujemy czy i jakie przerwania będą wykorzystywane przez program. Tę, pierwszą fazę działania naszego programu ukazuje poniższy listing, którego niektóre fragmenty zostały opatrzone dodatkowym komentarzem:

```
Dim Forward As Byte          'deklaracja zmiennej "ruch w przód"
Dim Back As Byte             'deklaracja zmiennej "ruch do tyłu"
Dim Left As Byte             'deklaracja zmiennej "skręt w lewo"
Dim Right As Byte            'deklaracja zmiennej "skręt w prawo"
Dim Backleft As Byte         'deklaracja zmiennej "skręt do tyłu w lewo"
Dim Backright As Byte        'deklaracja zmiennej "skręt do tyłu w prawo"
Dim Turnleft As Byte         'deklaracja zmiennej "obrót w lewo"
Dim Turnright As Byte        'deklaracja zmiennej "obrót w prawo"
Dim Sstop As Byte            'deklaracja zmiennej "stop"
Dim Command As Byte          'deklaracja zmiennej określającej numer wysłanej komendy
Dim Subaddress As Byte       'deklaracja zmiennej określającej adres urządzenia
Dim Aux As Byte              'deklaracja zmiennej określającej funkcję dodatkową
Dim New As Bit               'deklaracja zmiennej zawiadamiającej o odebraniu kodu RC5
Dim Licznik As Byte          'deklaracja zmiennej pomocniczej
Dim Flag As Bit              'deklaracja zmiennej pomocniczej
Dim Count As Long            'deklaracja zmiennej określającej czas oczekiwania na komendę RC5
```

Dodatkowy komentarz:
przewidując, że zmienna COUNT może mieć dużą wartość, deklarujemy ją jako LONG. Wartość zmiennej może teraz zawierać się w przedziale od 0 do 2 147 483 647.

```
Declare Sub Aux              'deklaracja podprogramu obsługującego funkcję dodatkową
P3 = 0 : P1 = 255 : Set P3.2 'ustawienie portów w stanach początkowych
```

Dodatkowy komentarz:
Podanie polecenia P[x] = Z powoduje wysłanie na wyjścia określonego portu binarnej reprezentacji liczby Z. A zatem, wszystkie wyjścia portu P3 znalazły się w stanie niskim, a wszystkie wyjścia portu P3 w stanie wysokim. Spowodowało to wyłączenie wszystkich urządzeń dołączonych do procesora. Jedynie wejście P3.2 zostało ustawione w stan wysoki, aby umożliwić odebranie transmisji z pilota. Polecenie SET ustawia na wybranym wyjściu stan wysoki, a polecenie RESET - stan niski.

```
Sound P1.0 , 500 , 1000      'generacja dźwięku sygnalizacyjnego
```

Dodatkowy komentarz:
Język MCS Basic dysponuje specyficznym poleceniem umożliwiającym generowanie prostych dźwięków, o niezbyt dokładnie określonej częstotliwości. Wydanie polecenia: SOUND [pin procesora], [czas trwania], [częstotliwość] powoduje wysłanie na wskazane wyprowadzenie procesora ciągu impulsów prostokątnych o zadanej częstotliwości. Szerzej omówimy to polecenie na jednej z następnych lekcji, ale już teraz warto wiedzieć, że nie możemy oczekiwać po nim generowania stabilnej i stałej częstotliwości i może być ono wykorzystywane jedynie do wytwarzania prostych dźwięków sygnalizacyjnych.

```
D1 Alias P1.7                'nadanie nowych nazw wyprowadzeniom procesora
D2 Alias P1.6                'nadanie nowych nazw wyprowadzeniom procesora
D3 Alias P1.5                'nadanie nowych nazw wyprowadzeniom procesora
D4 Alias P1.4                'nadanie nowych nazw wyprowadzeniom procesora
D5 Alias P1.3                'nadanie nowych nazw wyprowadzeniom procesora
D6 Alias P1.2                'nadanie nowych nazw wyprowadzeniom procesora
```

Dodatkowy komentarz:
Polecenie ALIAS jest przejawem iście dekadenckiego zamiłowania do wygody i komfortu pracy. Nie robi ono niczego innego w programie, poza tym, że po jego wydaniu możemy używać dwóch nazw wyprowadzenia procesora lub zmiennej. A zatem, po wydaniu polecenia: D1 Alias P1.7 staje się obojętne, czy wyprowadzenie P1.7 procesora nazywać będziemy nadal P1.7 czy też D1. Natomiast z punktu widzenia programisty jest znacznym udogodnieniem to, że nie musi już więcej sprawdzać na schemacie, do jakiego wyprowadzenia procesora dołączone jest każde z urządzeń (w naszym przypadku diody LED). Polecenie ALIAS jest tylko instrukcją dla kompilatora i jego stosowanie nie zwiększa długości kodu wynikowego.

```
Config Timer1 = Timer , Gate = Internal , Mode = 2 'konfigurowanie timera 1
```

Dodatkowy komentarz:
Korzystaniu z timerów poświęcona będzie w najbliższej przyszłości cała lekcja. Jest to bardzo obszerny i bardzo ważny temat, a na razie podaję Wam tylko najważniejsze informacje:

Polecenie Config Timer1 = Timer oznacza, że wewnętrzny licznik zawarty w strukturze procesora będzie pracował jako timer, czyli zliczał impulsy od zadanej liczby w dół. Po doliczeniu do zera timer zgłasza PRZERWANIE, czyli żądanie zaprzestania wykonywania przez procesor jakichkolwiek czynności i wykonanie instrukcji określonej poleceniem ON TIMER [0 lub 1].

Dodatkowe plecenie: Gate = Internal instruuje kompilator, że licznik wbudowany w procesor będzie zliczał impulsy pochodzące z zegara systemowego. Częstotliwość tych impulsów będzie równa częstotliwości oscylatora kwarcowego podzielonej przez 12.

Dodatkowe polecenie: Mode = 2 powoduje, że timer po osiągnięciu stanu 0 i wygenerowaniu przerwania zostanie natychmiast ponownie uruchomiony i rozpocznie zliczanie od tej samej, co uprzednio wartości. Tryb Mode = 2 jest wyjątkowo wygodny, ponieważ timer nie wymaga żadnej dodatkowej obsługi i pracuje niejako w "tle", jedynie zgłaszając upływ czasu za pomocą przerwań.

```
On Timer1 Timerint        'podprogram "Timerint" będzie obsługiwał przerwanie z timera
Load Timer1 , 255         'załadowanie rejestru timera początkową wartością
```

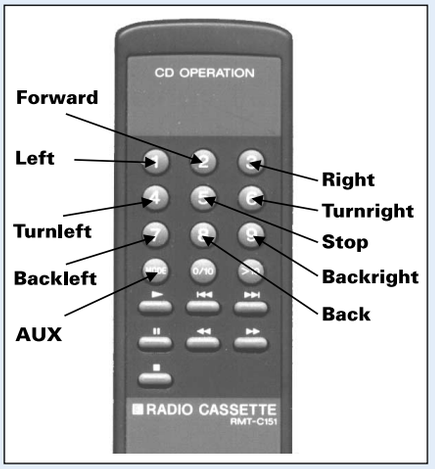
Dodatkowy komentarz:
Wydanie polecenia: Load timer [0,1] , [wartość] powoduje załadowanie do rejestru timera zadanej liczby, w przypadku pracy w trybie MODE =2 nie większej niż 255. Po uruchomieniu timera jego cykl pracy będzie wynosił 255 impulsów zegara systemowego. Teoretycznie w przypadku stosowania rezonatora kwarcowego 12MHz jeden taki cykl powinien trwać 0,000255 sekundy, w praktyce będzie trochę dłuższy. Do tematu timerów powrócimy na jednej z najbliższych lekcji.

```
On Int0 Receiverc5        'podprogram "Receiverc5" będzie obsługiwał przerwanie INT0
Enable Int0               'zezwolenie na korzystanie z przerwania INT0
Enable Interrupts         'zezwolenie na korzystanie z przerwań jako takich
Enable Timer1             'zezwolenie na korzystanie z przerwania z timera 1
```

W tym momencie program kończy przygotowania do pracy, polegające na określeniu konfiguracji sprzętowej i deklaracji stosowanych zmiennych i podprogramów. Następna część programu będzie składać się z szeregu małych pętli programowych. W każdej z nich program będzie starał się uzyskać informację, jaki kod odebrany z pilota będzie powodował określoną reakcję pojazdu. Z powodu ograniczonej ilości miejsca omówimy tylko pierwszą pętle, pamiętając, że pozostałe zbudowane są praktycznie identycznie, a różnica polega tylko na rejestrowaniu za każdym razem innej komendy. Oczywiście, nie ma najmniejszego znaczenia, jakiemu klawiszowi pilota podporządkujemy określoną komendę. Osobiście polecam wykorzystanie do sterowania pojazdem klawiszy numerycznych (**rysunek 3**), o ile oczywiście nasz pilot je posiada.

Po uzyskaniu informacji o kodzie, któremu podporządkowana jest zmienna FORWARD powodująca ruch pojazdu do przodu, program przechodzi do następnej pętli, w której próbuje uzyskać informacje o kodzie pilota, którego wysłanie ma powodować ruch pojazdu do tyłu. Tego fragmentu programu nie będziemy komentować, ponieważ zbudowany jest identycznie jak poprzedni i jak 8 następnych.

Rys. 3




```
Do                        'początek pętli programowej
Reset D4 : Reset D3       'włączenie diod D4 i D3, co jest żądaniem wysłania kodu powodującego
                                                    'ruch pojazdu do przodu
If New = 1 Then           'jeżeli odebrany został poprawny kod RC5 to:
   Disable Int0           'chwilowe zawieszenie obsługi przerwania INT0
   Forward = Command      'odebrane polecenie będzie powodować ruch pojazdu do przodu
   New = 0                'wyzerowanie znacznika odebrania kodu
   P1 = 0                 'wymuszenie stanu "0" na wszystkich wyjściach portu P1, a tym
                          'samym  włączenie wszystkich diod LED. Jest to sygnał
                                          potwierdzający   'odebranie żądanej komendy
   Wait 1                 'czekaj 1 s.
```

Dodatkowy komentarz:
Plecenie:

WAIT [s.]

powoduje zwieszenie działania programu na podany okres. Oczekiwanie może zakończyć się po upływie tego czasu oraz po wystąpieniu sygnału przerwania zewnętrznego lub pochodzącego z timera. Maksymalny czas może wynosić 255 s.
Podobną rolę pełni dyrektywa:

WAITMS [ms.]

UWAGA. Funkcja WAIT i WAITMS nie odmierzają czasu zbyt precyzyjnie i nie mogą być stosowane do dokonywania precyzyjnych pomiarów.

```
P1 = 255          'wyłączenie diod LED, których chwilowe włączenie było dla 'użytkownika sygnałem
                                                    o  odebraniu kodu
   Enable Int0    'ponowne zezwolenie na obsługę przerwania
   Exit Do        'wyjście z pętli programowej
```

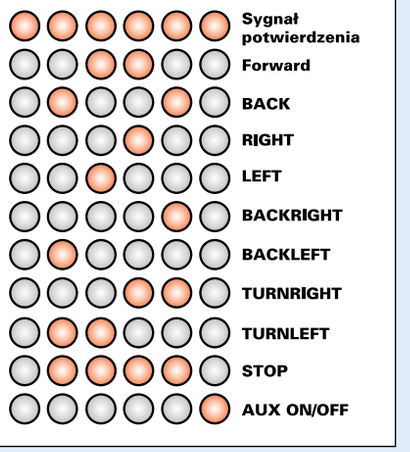
Dodatkowy komentarz:
Polecenie:
EXIT DO powoduje natychmiastowe wyjście z pętli programowej określonej poleceniami DO .... LOOP. Po wyjściu z pętli program rozpoczyna dalszą działalność od pierwszej linii występującej po "LOOP"

```
 End If  'koniec uwarunkowania poprawnego odebrania transmisji RC5
Loop     'zamknięcie pętli
```

```
Do
  If New = 1 Then
    Disable Int0
    Back = Command
    New = 0
    P1 = 0
    Wait 1
    P1 = 255
    Enable Int0
    Exit Do
  End If
Loop
```

Każde żądanie wysłania określonej komendy poprzedzone jest włączeniem odpowiadającej jej kombinacji diod LED. Dla ułatwienia wszystkie te kombinacje zostały pokazane na **rysunku 4**.

Rys. 4



Po zakończeniu zdobywania informacji o potrzebnych mu kodach, program wychodzi znowu "na prostą" i realizuje kilka funkcji przed wejściem w kolejną pętlę programową, a przede wszystkim 10 błyskami diod LED zawiadamia o zakończeniu pierwszego zadania :

```
For Licznik = 1 To 10   'wykonaj poniższe instrukcje 10 razy
P1 = 255                'wyłącz wszystkie diody LED dołączone do portu 1
Waitms 200              'zaczekaj 200 ok. msek
Sound P1.0 , 100 , 1000 'wygeneruj krótki dźwięk
P1 = 0                  ' włącz wszystkie diody dołączone do portu 1
Waitms 200              'zaczekaj 200 msek
Next Licznik            ' powtórz powyższe czynności
Reset P3.1 : Reset P3.3 : Reset P3.4 : Reset P3.5 : P1 = 255 ' wyłącz diody, wyłącz silniki
Enable Interrupts       'zezwalam na obsługę przerwań
On Int0 Receiverc5      'w przypadku wystąpienia zgłoszenia przerwania (odebrania 'RC5 skocz
                                    do podprogramu "Receiverc5"
```

```
Enable Int0             'zezwolenie na obsługę przerwania INT0
New = 0                 'wyzerowanie znacznika odebrania kodu RC5
Command = 255           'zmienna COMMAND ma mieć jak na razie wartość 255
Priority Set Int0       'przerwanie pochodzące od odbiornika RC5 ma 'pierwszeństwo przed
                        przerwaniem pochodzącym z timera.
                        'uruchomienie timera 1
```

Dodatkowy komentarz:

Stosowanie przerwań jest jednym z największych udogodnień dla programisty, ale może też być źródłem poważnych kłopotów. W przypadku stosowania dwóch lub więcej źródeł przerwań, bezwzględnie konieczne jest ustalenie, które z nich ma pierwszeństwo i które zostanie pierwsze obsłużone. W naszym przypadku zadecydowaliśmy, że zawsze pierwszeństwo ma przerwanie pochodzące z odbiornika RC5

```
Start Timer1
```

Dodatkowy komentarz:

Pamiętamy, że TIMER0 został skonfigurowany jako licznik zliczający w dół od liczby 255. Po każdym osiągnięciu przez timer stanu "0" zgłaszane jest przerwanie, po czym timer automatycznie wpisuje do swojego rejestru zadaną uprzednio wartość i rozpoczyna zliczanie od początku. Każde przerwanie zgłoszone przez timer powoduje skok do podprogramu: "TIMERINT", z którego działaniem zapoznamy się za chwilę.

```
Count = 0               'wyzerowanie licznika COUNT (patrz komentarz dalej)
```

Po zakończeniu rejestrowania komend i po wykonaniu czynności przygotowawczych program wchodzi w pętlę, w której pozostanie aż do momentu wyłączenia zasilania. Jedynym wytchnieniem dla zmęczonych krasnoludków biegających nieustannie we wnętrzu procesora może być ewentualne chwilowe zawieszenie pracy systemu, którego działanie omówimy w dalszej kolejności.

Każda odebrana komenda pochodząca z nadajnika kodu RC5 jest przez program analizowana. Jeżeli komenda taka odpowiada jednemu z zarejestrowanych poleceń, to program podejmuje określone działania, kierując ruchem pojazdu zgodnie z naszymi życzeniami. Pętla programowa zbudowana jest z szeregu uwarunkowań analizujących odebraną komendę i wykonujących odpowiadające jej czynności. Opiszemy tylko dwa pierwsze uwarunkowania, pamiętając, że pozostałe zbudowane są bardzo podobnie.

Do omówienia pozostały nam już tylko dwa podprogramy - procedury analizowania odebranego kodu RC5 i zawieszania pracy programu po dłuższym okresie bezczynności. Pierwszy podprogram RECEIVERC5 został już szczegółowo opisany w ćwiczeniu 2. Zwróćmy jedynie uwagę na fakt, że przy każdym odebraniu poprawnej transmisji RC5 zerowana jest zmienna COUNT, co ma ogromne znaczenie dla poprawnej pracy programu.

```
Receiverc5:
Disable Int0
Getrc5(subaddress , Command)
If Command < 63 Then
  New = 1
  Count = 0
'zerowanie zmiennej COUNT
  End If
Enable Int0
Return
```

Ostatnim podprogramem wartym szczegółowego omówienia jest procedura, której zadaniem jest radykalne zmniejszenie mocy pobieranej przez naszą zabawkę, w momentach krótkich przerw w zabawie.

Na samym początku działania naszego programu został wydany ciąg poleceń konfigurujących i uruchamiających timer systemowy TIMER0. Przypomnijmy je sobie:

```
Config Timer1 = Timer , Gate = Internal , Mode = 2
```

A następnie:

```
Enable Timer1
```

I po zarejestrowaniu komend:

```
Load Timer1, 255
Start TIMER1
```

Od momentu wydania ostatniej instrukcji zostaje uruchomiony timer sprzętowy procesora, a jego działanie nie będzie już w naszym programie zatrzymane. W momencie osiągnięcia przez timer stanu 0 generowany jest sygnał przerwania, po czym timer rozpoczyna ponowne zliczanie w dół od zadanej poleceniem LOAD TIMER liczby. Przy każdym zgłoszeniu przerwania wykonywany jest skok do poniższej procedury, nazwanej TIMERINT (nie jest to element składni MCS BASIC, ale po prostu nazwa):

```
Timerint:
  Incr Count
          'zwiększ wartość zmiennej COUNT
```

Dodatkowy komentarz:
Polecenie INCR [zmienna] powoduje zwiększenie wartości zmiennej o 1. Podobne działanie miałoby polecenie: Zmienna = zmienna +1,
ale polecenie INCR wykonywane jest przez procesor znacznie szybciej
Poleceniem o odwrotnym działaniu do INCR jest DECR, które powoduje zmniejszenie wartości zmiennej o 1.

```
If Count = 50000 Then 'jeżeli zmienna COUNT przyjęła 'wartość 50000, to:
P1 = 255
          'wyłącz wszystkie diody LED
Reset P3.1 : Reset P3.3 : Reset P3.4 : Reset P3.5
          'wyłącz silniki
For Licznik = 1 To 5    'pięciokrotnie wykonaj poniższe 'instrukcje
Sound P1.0 , 500 , 2000
          'wygeneruj sygnał akustyczny
Waitms 200
          'zaczekaj 200ms.
Next Licznik
Idle      'wprowadź procesor w stan
                'ograniczonego poboru mocy
  End If
          'koniec uwarunkowania
Return    'powrót do programu głównego,
            o ile 'warunek nie został spełniony
```

Podprogram TIMERINT realizuje następujące zadania:

1. Po każdorazowym wystąpieniu przerwania timera1 wartość zmiennej COUNT zwiększana jest o 1. Pamiętajmy jednak, że każde odebranie transmisji z pilota powoduje zerowanie tej zmiennej, tak że podczas normalnej eksploatacji zabawki nie może ona osiągnąć zbyt wielkiej wartości.

2. O ile jednak przez pewien czas nie nadajemy jakichkolwiek komend z pilota, to zmienna COUNT osiągnie zadaną w linii uwarunkowania wartość: 50000. Konsekwencje tego faktu będą następujące:
- zostaną wyłączone wszystkie diody LED
- wyłączone zostaną silniki, o ile przed spełnieniem uwarunkowania były włączone
- zostanie wygenerowany pięciokrotny sygnał ostrzegawczy

```
Do                              'początek pętli programowej

If Command = Forward Then       'jeżeli została odebrana komenda "FORWARD", to:
Reset P3.3 : Reset P3.5         'ustaw stan niski na wyjściach powodujących pracę silnika do tyłu
Set P3.1 : Set P3.4             'włącz obydwa silniki w tym samym kierunku: do 'przodu
Reset D3 : Reset D4 : Set D2 : Set D5    'włącz odpowiadającą odebranej komendzie 'kombinację
                                                                    diod LED
Disable Timer1                  'chwilowo zawieś obsługę przerwania timera 1
Sound P1.0 , 100 , 1000         'wygeneruj sygnał akustyczny
Enable Timer1                   'ponownie wydaj zezwolenie na obsługę przerwania 'timera 1
Command = 255                   'zmień wartość "COMMAND" na nieakceptowaną 'przez programy
                                                                    analizujące.
End If                          'koniec uwarunkowania

If Command = Back Then          'jeżeli została odebrana komenda "FORWARD", to:
Reset P3.1 : Reset P3.4         'ustaw stan niski na wyjściach powodujących pracę 'silnika do przodu
Set P3.3 : Set P3.5             'włącz obydwa silniki w tym samym kierunku: do tyłu
Reset D2 : Reset D5 : Set D3 : Set D4    'włącz odpowiadającą odebranej komendzie 'kombinację
                                                                    diod LED
Disable Timer1                  'chwilowo zawieś obsługę przerwania timera 1
Sound P1.0 , 100 , 1000         'wygeneruj sygnał akustyczny
Enable Timer1                   'ponownie wydaj zezwolenie na obsługę przerwania 'timera 1
Command = 255                   'zmień wartość "COMMAND" na nieakceptowaną 'przez podprogramy
                                                                    analizujące.
End If                          'koniec uwarunkowania
```

- procesor zostanie wprowadzony w stan zmniejszonego poboru mocy, w którym będzie pozostawał do czasu odebrania następnej komendy z pilota lub do wyłączenia zasilania.

Podczas "drzemki" procesora pobór prądu spada do ok. 3 mA i dostęp do portów zostaje zablokowany. Warto jednak pamiętać, że nie tylko zawartość wszystkich rejestrów zostaje zachowana, ale także aktywne pozostają timery i wejścia przerwań. Macie zatem temat do przemyślenia: jak będzie się zachowywać nasza zabawka podczas przedłużającej się drzemki?

I jeszcze jeden temat do przemyśleń: wykonaliśmy i oprogramowaliśmy zabawkę, która realizuje dość skomplikowane funkcje. Potrafi się uczyć, wykonywać przekazywane jej polecenia, a nawet w nudnych momentach uciąć sobie drzemkę. Do zbudowania tego urządzenia potrzebny był zaledwie jeden układ scalony: procesor i garstka tanich i powszechnie dostępnych elementów dyskretnych. Czy można by było wykonać takie urządzenie stosując "klasyczne" metody? Oczywiście, że tak, ale spróbujcie narysować schemat takiego układu zbudowanego z wykorzystaniem kostek TTL czy 4000!

W ciągu kilku chwil dokonaliśmy gigantycznego skoku: z początku lat osiemdziesiątych zostaliśmy przeniesieni w świat nowoczesnej techniki mikroprocesorowej. Chyba było warto!

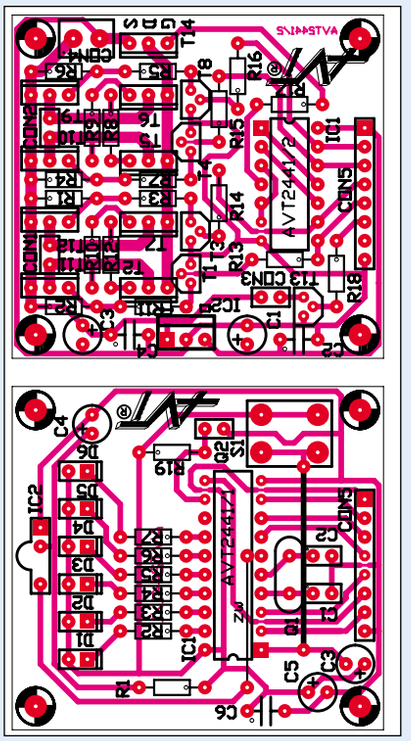
Montaż i uruchomienie

Na **rysunku 5** zostało pokazane rozmieszczenie elementów na dwóch płytkach obwodów drukowanych wykonanych na laminacie jednostronnym. Nie sądzę, aby ktokolwiek z Was miał jakiekolwiek problemy ze zmontowaniem tych dwóch malutkich płytek i połączeniu ich ze sobą za pomocą szeregu goldpinów (z możliwością rozłączania płytek) lub po prostu odcinków srebrzanki. Szczegóły montażu "kanapki" złożonej z dwóch płytek widoczne są na zdjęciach. Komentarza wymaga jedynie sprawa stabilizatora napięcia 7805, którego wykorzystywanie jest opcjonalne i uzależnione wyłącznie od napięcia zasilania układu wykonawczego. W układzie prototypowym, zasilanym z czterech baterii napięciem 6V, stabilizator nie był stosowany (procesor 89C2051 może pracować przy tym napięciu!). Jeżeli jednak do zasilania silników będziecie potrzebować wyższego napięcia, np. 12V, to należy wlutować w płytkę stabilizator 7805.

Montaż płytek to i tak fraszka w porównaniu z wykonaniem układu napędowego. Tu jednak trudno mi podać Wam jakieś gotowe rozwiązanie. Poszukajcie wśród zniszczonych zabawek młodszego rodzeństwa, w sklepach z "bublami", może zdemolujecie jakiś stary zegar....

Zbigniew Raabe
e-mail: **zbigniew.raabe@edw.com.pl**

Rys. 5 Schemat montażowy



Wykaz elementów - układ wykonawczy

Kondensatory
C1, C3 ... 100µF
C2, C4 ... 100nF

Rezystory
R1 - R12 ... 300 Ω
R13 - R16 ... 22kΩ
R17 ... 100 Ω
R18 ... 3,3Ω

Półprzewodniki
IC1 ... 4011
IC2 ... 7805
T1, T3, T4, T8, T13 ... BC548
T2, T5, T6, T7 ... BD139
T9 - T12 ... BD140
T14 ... BUZ10

Pozostałe
CON4 ... ARK2

Wykaz elementów - układ sterujący

Kondensatory
C1, C2 ... 33pF
C3 ... 1µF
C4, C5 ... 100µF
C6 ... 100nF

Rezystory
R1 ... 100
R2 - R7 ... 560

Półprzewodniki
D1 - D6 ... LED
IC1 ... 89C2051
IC2 ... TFMS5360

Pozostałe
Q1Q ... rezonator kwarcowy 11,059MHz
S1 ... przycisk microswitch
8 goldpinów
8 pinłącze szufladkowe
piezo A10
złącze ARK2 3.5 mm

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-2441