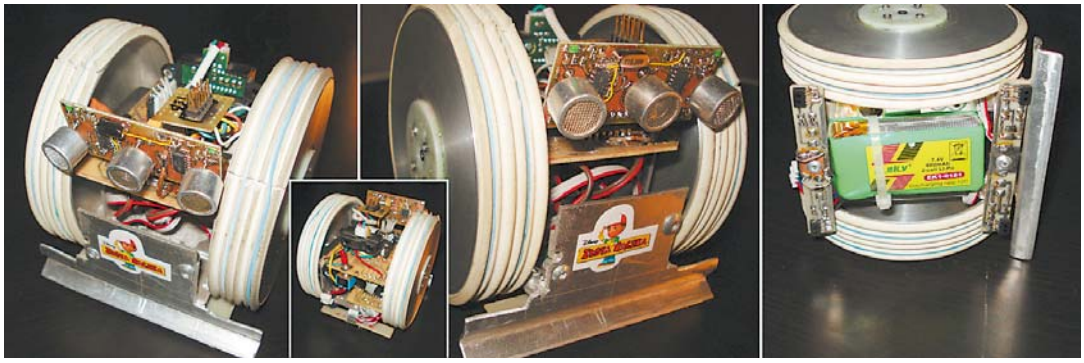
kit 2935 AVT



Robot dla każdego, czyli także dla Ciebie

część 1

Witam! Na imię mam Marek. Elektroniką interesuję się od zawsze :-). Robotami też – to od lat moja pasja. Chcąc pozostawić coś po sobie potomnym, szukałem okazji… i nadarzyła się ona poprzez ofertę Redakcji EdW. A mianowicie otrzymałem propozycję, żebym napisał obszerny artykuł o budowie robota. Po kilku e-mailach z redaktorem Piotrem Góreckim zaczęła powstawać seria, której pierwszy odcinek właśnie masz, drogi Czytelniku, przed oczami.

Na początek chciałbym troszeczkę przybliżyć moją osobę i pokazać kilka swoich konstrukcji. Otóż robotyką zainteresowałem się kawałek czasu temu, dokładniej w okolicach roku 1996. Wtedy to obejrzałem na video film pt. „Robocik", następnie legendarny film pod tytułem „Krótkie Spięcie". Zarówno pierwsza, jak i druga część, wywarły na mnie mocne i niezatarte wrażenie. Pierwszym robotem, jaki zbudowałem, była moja praca dyplomowa w technikum Elektronicznym w Kielcach. Robot posiadał kilka stopni swobody i sterowany był bezprzewodowo z komputera PC. Oprogramowanie powstało w środowisku Turbo Basic. Niestety nie posiadam już kolorowych zdjęć tego robota. **Fotografia 1** to jedyne ujęcie, które mi pozostało. Robot ten tak naprawdę nie jest niczym szczególnym, patrząc na niego z poziomu obecnej elektroniki. Ale patrząc z poziomu roku 1997, był zaawansowaną konstrukcję, zwłaszcza jak na ucznia technikum. Otrzymał wyróżnienie w Turnieju Młodych Mistrzów Techniki stopnia ogólnopolskiego. Robot ten posiadał napęd w postaci dwóch silników od wycieraczek i koła podpierającego. Ręka (manipulator) posiadała 5 silników. Głowa dwa. Wbudowana kamera, mikrofon i głośniki zapewniały komunikację podczas sterowania zdalnego.

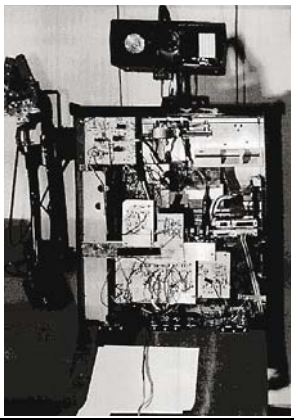
Nie był to koniec. Później budowałem różne mniej lub bardziej udane konstrukcje, których dokumentacji już nie posiadam. Filmiki z zawodów robotów sumo zaciekawiła mnie tą tematyką. Skutkiem był kontakt z kolegą Karolem z koła robotyków KONAR. I poznanie wspaniałego robota kolegi Karola (był to robot klasy nanosumo). Wtedy wpadłem na szalony pomysł zbudowania swojego robocika klasy nanosumo. Po wielu bojach powstał model, pokazany na **fotografii 2**.

Jak widać jest maleńki. Wymiary robotów klasy nanosumo to zaledwie 25x25x25mm, a dopuszczalna masa 25g! Napęd stanowią silniki wyciągnięte z mikroautek. Czujniki otoczenia to wyciągnięte z magnetowidów Sharp sensory odbiciowe. Przód stanowi czujnik IS471F kupiony od kolegi. Akumulatorek firmy Kokam.

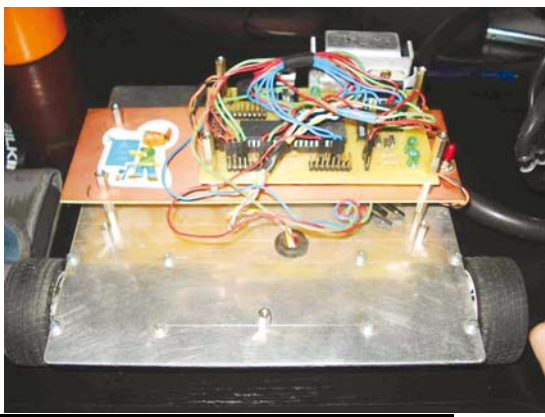
Następnie pojawiła się konstrukcja robota klasy minisumo. Dla przypomnienia: w robocie minisumo wielkość podstawy nie może przekroczyć 10x10cm, a masa startowa nie może przekroczyć 500g. Robot widoczny na **fotografii tytułowej** to moja pierwsza konstrukcja tego typu. Czujniki koloru (podłoża) są typu TCRT5000, a robot widzi „świat", korzystając z lekko zmodyfikowanego sonaru opracowanego przez koło robotyków KONAR. Napęd stanowią dwa zmodyfikowane serwa modelarskie. Jak zmodyfikowane, o tym będzie o tym mowa w dalszych częściach cyklu. Następna konstrukcja to robot, który zaprojektowałem (który powstaje nadal) z myślą o moim 3-letnim synku, dlatego nazywa się „robot Krzysia" – **fotografia 3**. Jest to prosta platforma, napędzana 2 silnikami firmy Tamiya, które można było kupić na allegro. Na fotografii konstrukcja widoczna jest od frontu. Pod płytką znajduje się akumulator żelowy 6V – niewidoczny na tej fotografii, bo wtedy jeszcze go tam nie było.

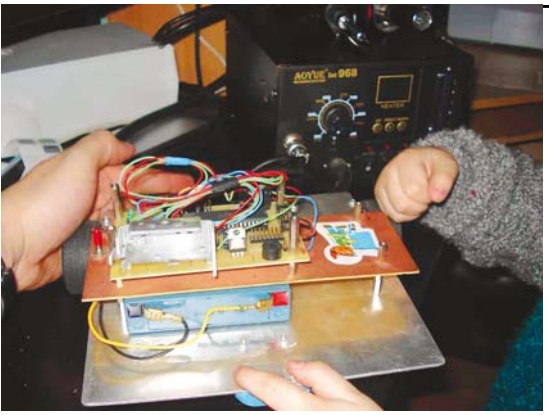
Fot. 2

Fot. 1

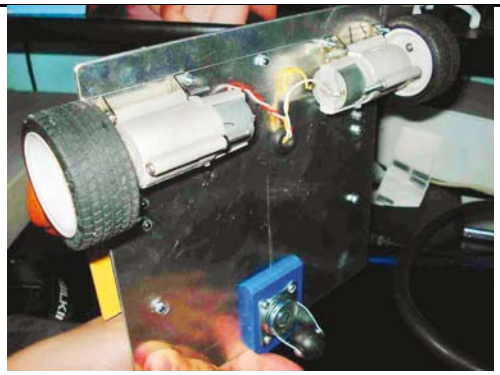


Fot. 3

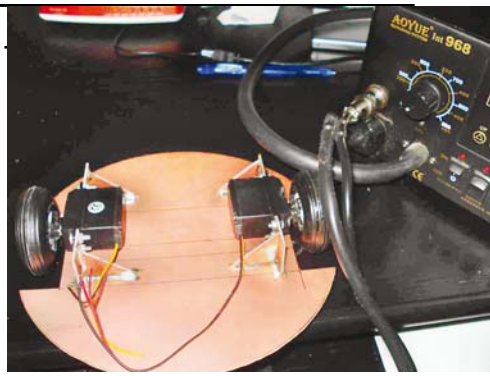




Fot. 4



Fot. 5



Fot. 6

Do tego dochodzi odbiornik IR i pilot RC5. Jest to konstrukcja, która w stanie pokazanym na fotografii była po prostu zdalnie sterowaną zabawką. Inne ujęcie tej konstrukcji pokazane jest na **fotografii 4**. Natomiast na **fotografii 5** widoczne jest podwozie z silnikami. Wyraźnie widać dwa silniki z wbudowanymi przekładniami oraz koła firmy Tamiya. Trzecie kółko (obrotowe) pochodzi z Castoramy.

Następnie powstał robot, a w zasadzie podwozie robota, które nadal jest w trakcie rozwoju (brak czasu). Robot ten miał być moim pierwszym LF (line follower). Napęd stanowią dwa zmodyfikowane serwomechanizmy. Jak widać, całość konstrukcji wykonana jest z laminatu. Tak, z płytek drukowanych. Podocinanych i polutowanych ze sobą. Każda metoda budowy jest dopuszczalna :-) . Widok tej konstrukcji przedstawiony jest na **fotografii 6**. **Fotografia 7** przedstawia kilka detali mocowania serw.

W międzyczasie powstaje mój, jak dotąd najbardziej zaawansowany, robot. Konstrukcja oparta jest na stalowym stelażu wykonanym z kątowników 20x20 mm. Obudowę prezentuję na **fotografii 8**. Na pierwszy rzut oka jest to prosty z wyglądu i niezbyt estetyczny robot. Jednakże jego głównym składnikiem jest... przerobiony laptop, który ma pracować w środowisku Linux. A zastosowanie takiego rozwiązania podyktowane było koniecznością obsługi portów USB, analizy obrazu z kamerki internetowej, która jest widoczna tuż nad (wygaszonym) ekranem LCD i wyświetlaniem na tym ekranie dużych „emotikonek". Napęd to dwa silniki od wycieraczek samochodowych. Zasilanie: dwa akumulatory żelowe o pojemności 12Ah każdy. Reszta robota nadal jest w fazie budowy i nie ma tam jeszcze większości elektroniki.

W ramach pomysłu na robota, którego będę mógł budować gdziekolwiek, powstała konstrukcja, pokazana na **fotografii 9**. Robot napędzany jest dwoma silnikami z wbudowanymi przekładniami planetarnymi. Koła oraz przelotki mocujące zostały kupione w firmie Wobit (*www.mobot.pl*). Koła są bardzo fajne ze względy na dużą średnicę oraz bardzo dobrą przyczepność. Robot jak do tej pory, posiada wbudowany syntezator mowy („gadaczka" wykorzystująca moduł ISD), bargraf pokazujący stan naładowania akumulatora i sonar ultradźwiekowy, bazujący na konstrukcji KONAR-u. Algorytm robota na chwilę obecną pozwala omijać przeszkody, a jeżeli przeszkoda znajduje się na wprost, po prostu od niej odjeżdża, robi zwrot i kontynuuje jazdę do przodu. Druga wersja programu robi z niego tzw. „rzepa na psim ogonie": robot po prostu w miarę możliwości podąża za osobą lub przedmiotem, starając się zachować stałą odległość.

Ale, ale… niniejszy artykuł miał mówić o nowej konstrukcji, o robocie którego zbudujemy razem … Zakończmy wiec wstęp i przejdźmy do sedna sprawy. Zaznaczam jednak, że nie jestem specjalistą w pisaniu, więc proszę o wyrozumiałość.

Cel cyklu

Celem tego cyklu jest, by każdy Czytelnik potrafił samodzielnie zbudować małego robota. Nie chodzi jednak o robota człekokształtnego, tylko o niewielki pojazd, obdarzony pewną miara inteligencji, tak jak konstrukcje zaprezentowane na wcześniejszych fotografiach.

Wbrew pozorom, budowa takiego robota nie jest bardzo trudna. Mogą się tego podjąć nawet osoby mało doświadczone i młode. Oczywiście można kupić, czy to gotowego robota, czy gotowe moduły, czy kluczowe podzespoły, jednak co ciekawe i bardzo ważne, koszty wcale nie muszą być wysokie! Otóż można wykorzystać wiele części z odzysku, a całą elektronikę, łącznie z płytkami drukowanymi, wykonać samodzielnie. Oprócz obniżki kosztów da to ogromną satysfakcję z wykonania wszystkiego własnoręcznie.

Jednak ktoś, kto chciałby bez przygotowania zacząć taką budowę od zera, napotyka wiele problemów, pytań, wątpliwości i pułapek. Budowa nawet prostego robota to połączenie elektroniki, informatyki i mechaniki. Budowa robota to spore przedsięwzięcie, ponieważ trzeba trochę znać się i na elektronice, i na programowaniu, i na mechanice. W grę wchodzi wiele szczegółów i niestety niektórzy początkujący albo boją się tak złożonego zadania, albo popełniają istotne

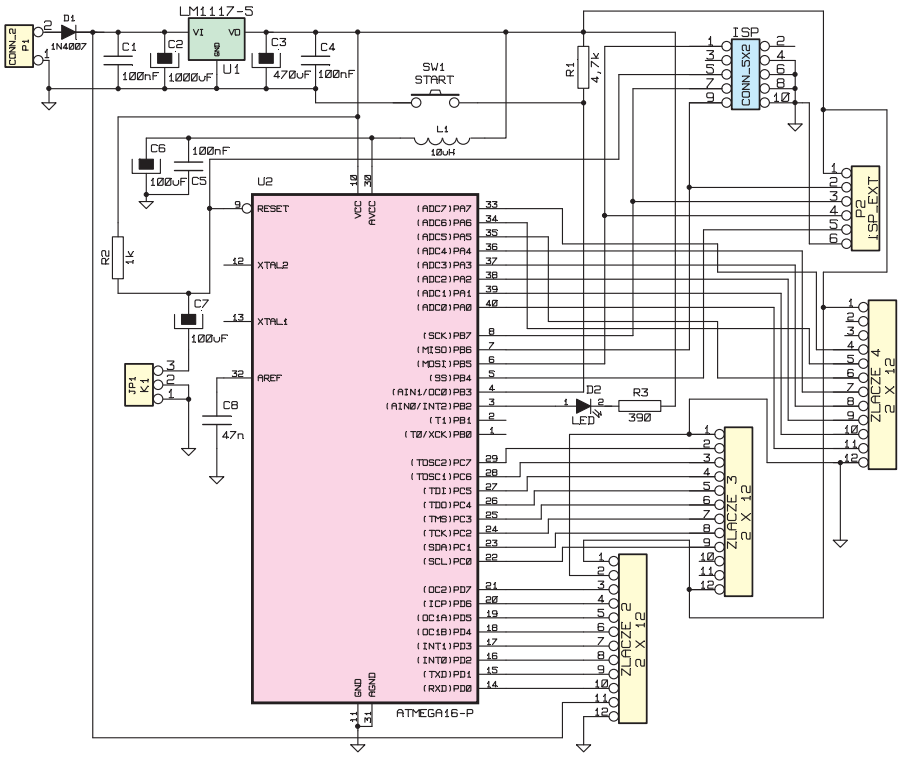
Fot. 7

Fot. 8

Fot. 9

błędy, które nie pozwalają osiągnąć pełnego sukcesu.

I właśnie niniejszy cykl ma pomóc, nie tylko początkującym, w prawidłowej realizacji robota od początku do końca. Nie jest to „jedynie słuszny" projekt. Nie ma to też być encyklopedyczny poradnik „how to", lecz koncept. Owszem, pokażę, jak się buduje konkretnego robota, ale zachęcam do samodzielności i do sprawdzania własnych pomysłów. Będę też podawał źródła zakupu i pozyskania części: gdzie co można kupić, z czego można wyciągnąć itd. I co bardzo ważne: postaram się pomóc i podpowiedzieć drogą e-mailową każdemu, kto będzie chętny przystąpić do grona konstruktorów robotów.



Rys. 1

Na pewno zauważycie, że mój cykl jest inny, niż typowe projekty w EdW. Celem tego odcinka jest wprowadzenie w zagadnienie, zachęcenie do podjęcia działań oraz zgromadzenie potrzebnych części i materiałów. W następnych odcinkach będziemy szczegółowo omawiać poszczególne etapy prac. A na koniec zapewne ogłosimy konkurs, żebyście mogli zaprezentować swoje osiągnięcia, związane z tym cyklem.

Tyle tytułem wprowadzenia i zgodnie z wcześniejszymi zapowiedziami przechodzimy do konkretów.

Elektronika

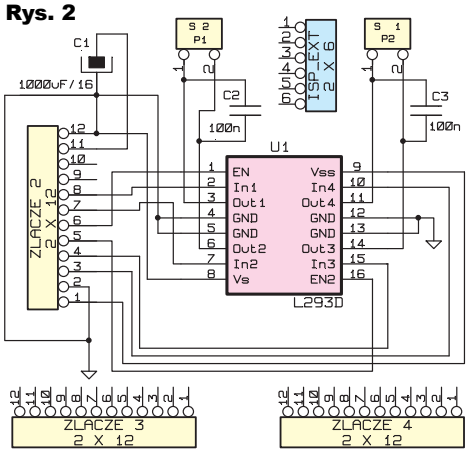
Być może myślałeś, że zaczniemy od kwestii mechanicznych. Proponuję inną kolejność. Otóż napęd jest specyficzny dla każdego konstruktora amatora bądź profesjonalisty (uzależniony jest też od możliwości technologicznych i zasobów portfela). Dlatego konstrukcją mechaniki robota zajmiemy się pod koniec cyklu i tam też omówimy kilka możliwych wariantów układu napędowego.

Natomiast w pierwszej części postaram się opisać kilka podstawowych tematów związanych z elektroniką. Podpowiem co, gdzie i jak. Zaczniemy tworzyć płytki drukowane do dwóch podstawowych części naszej konstrukcji. Będą to mianowicie:

- „mózg" robota,
- sterownik silników.

Dzięki tym płytkom będziemy mieli możliwość uruchomienia oprogramowania na komputerze i zabawy z programem sterującym robota (jego „inteligencją").

Rys. 2



Zaproponowane tutaj układy elektroniczne mogą stanowić bazę do różnorakich konstrukcji. Zaprojektowane są tak, by pasowały do konstrukcji praktycznie dowolnego robota. Będzie można na nich odpalić robota klasy sumo, minisumo, follow the line, freestyle. Zgodnie z tym założeniem, nie możemy przekroczyć wymiarów 10cm x 10cm (wymagania minisumo).

Realizacja części elektronicznej naszego robota obejmuje:

1. ***Wykonanie lub zakup płytek drukowanych***
2. ***Montaż układów na płytkach***
3. ***Zaprogramowanie procesora***

Co będzie nam do tego potrzebne? Do prac potrzebujemy narzędzi. Jakich? No cóż, na pewno będzie potrzebna lutownica, jakiś miernik, ucinaczki (cążki), cyna, pinceta. Omówmy teraz kolejne etapy pracy nad elektroniką.

Wykonanie lub zakup płytek drukowanych

Płytki drukowane proponowanej przeze mnie wersji można będzie zamówić w AVT lub wykonywać samodzielnie. Zachęcam do tej drugiej opcji! Jeżeli robimy płytki samodzielnie, dodatkowo potrzebować będziemy odczynników chemicznych i narzędzi do obróbki mechanicznej (między innymi np. wiertarki). Zapewne przyda nam się też poradnik, który znajduje się pod tym adresem: *www.ensyst.pl/szablon/files/METODA_ZELAZKOWA.pdf.*

Oczywiście potrzebne są pliki projektów dwóch płytek. Pliki dostępne są w Elportalu wśród materiałów dodatkowych do tego numeru EdW, ale dla ułatwienia można tam dotrzeć także, wpisując adres: *www.elportal.pl/robot.*

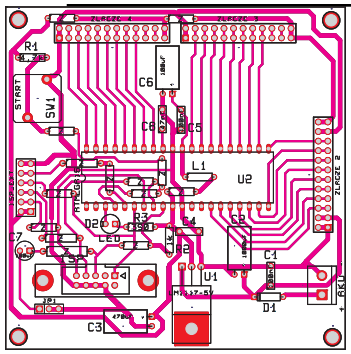
Znajdziesz tam, Czytelniku, gotowe do druku rysunki ścieżek, warstwy opisu oraz schematy w formacie PDF. Ale umieściłem też tam projekty źródłowe płytek, które jeśli chcesz, możesz modyfikować według swoich upodobań. Otóż płytki projektowałem w darmowym programie KiCad, który można pobrać z tego adresu: *http://kicad.sourceforge.net/wiki/index.php/PL:Main_Page.*

Części (laminat i chemikalia) niezbędne do wykonania płytek możemy kupić praktycznie w każdym sklepie elektronicznym. Między innymi: *www.sklep.avt.pl, www.piekarz.pl, www.tme.pl*.

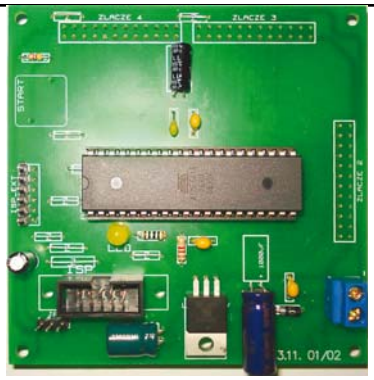
W każdym razie finałem tego etapu prac będą dwie płytki drukowane.

Budowa modułów elektronicznych

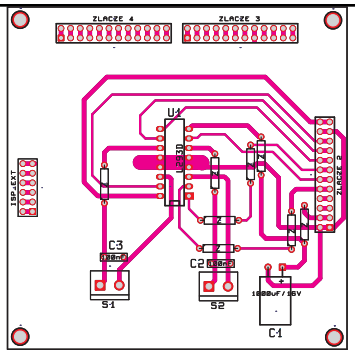
Schematy elektryczne modułu mózgu i sterownika silników pokazane są na **rysunkach 1** i **2**. Jak widać na rysunku 1, kluczowym elementem jest procesor Atmega16. Otoczony on jest rozmaitymi złączami i gniazdami. Jedno służy do programowania, pozostałe do komunikacji z innymi modułami. Oprócz tego na płytce umieszczone są standardowe obwody zasilania ze stabilizatorem U1. Proponuję wykorzystać stabilizator typu LDO, na przykład LM1117-5V. Dioda D1 też dla obniżenia strat mogłaby być diodą Schottky'ego, np.



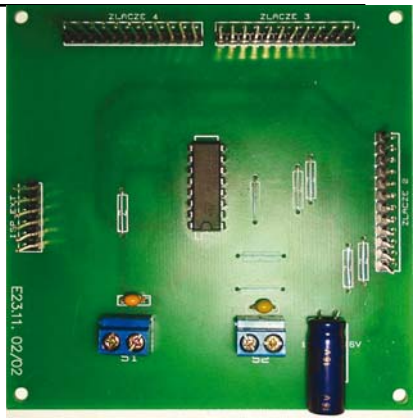
Rys. 3 Skala 50%



Fot. 10



Rys. 4 Skala 50%



Fot. 11

1N5817...5819. Przy zasilaniu z akumulatora 12V można zastosować stabilizator 7805 i diodę 1N4001...7

Na **rysunku 3** i **fotografii 10** pokazane są projekty płytki „mózgu". Jak widać, płytka jest dość prosta. Zawiera wprawdzie sporą ilość zworek (oznaczenie Z), lecz dzięki temu nie jest wymagana płytka dwustronna. A jednostronną łatwo wykonać we własnym zakresie. W naszej konstrukcji posłużymy się procesorem serii AVR, dokładniej Atmega16 i co ważne, w obudowie DIP. Zakładam bowiem, że nie każdy posiada sprzęt i umiejętności potrzebne do lutowania SMD – to ukłon w stronę mniej zaawansowanych. Do kompletu potrzebujemy kilku elementów, jak kondensatory, rezystory, złączki, trochę przewodów.

Na **rysunku 4** i **fotografii 11** znajdziesz płytkę sterownika silników. Do sterownika napędu wykorzystamy układ L293D – jeżeli planujemy mniejszego robota lub robota, który ma silniki o stosunkowo małym poborze prądu. Ważne jest, by kupić układ L293D. Literka D w jego nazwie oznacza, iż zawiera on już wbudowane diody zabezpieczające i upraszcza konstrukcję. Do tego kilka kondensatorów, troszkę przewodów :-). Pełny wykaz na końcu artykułu.

Czy nasz robot będzie konstrukcją opartą na dwóch, czy na czterech silnikach, nie ma to większego znaczenia. Do sterowania tego typu konstrukcji posługujemy się prostą metodą: silnik/i lewe i prawe zmieniają kierunek ruchu. Praca równoczesna powoduje jazdę w tył lub przód. Płytka sterownika powstała jako „kanapka" do pierwszej płytki „mózgu" - na pozór niepotrzebne, niepodłączone gniazda potrzebna są m.in. właśnie po to, żeby z modułów stworzyć „kanapkę". Dzięki temu nasz robot będzie konstrukcją dość zwartą. Lecz nie spowoduje to braku możliwości rozłożenia płytek i zastosowania połączeń przewodowych.

Montaż układów na płytkach

Przy kompletowaniu elementów należy wykorzystać szczegółowy wykaz zamieszczony na końcu artykułu.

Płytki procesora i sterownika należy polutować bardzo starannie, by uniknąć kłopotów z błędnym działaniem. Lutowanie zawsze zaczynamy od wlutowania elementów najmniejszych, takich jak zworki, rezystory itp. Na początek do umieszczenia zworek potrzebujemy zestawu narzędzi, jak na **fotografii 12**.

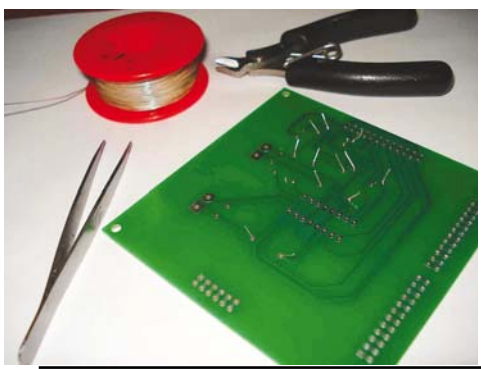
Jak widać, dobrze jest po umieszczeniu zworki na miejscu, lekko rozgiąć jej zakończenia. Dzięki takiemu zabiegowi nic nam nie wypadnie podczas lutowania. Następnie lutujemy i obcinamy zbędne wystające kawałki (są tu różne szkoły jedne mówią o innej kolejności lecz ja mam swoją opracowaną przez lata i na jej podstawie będę chciał przekazać swoją wiedzę). Do tego celu przyda nam się zestaw narzędzi, przedstawiony na **fotografii 13**.

Cóż, w dzisiejszych czasach, jeśli ktoś nie posiada klasycznej lutownicy „kolbowej", to prędzej czy później takową mieć musi. Przy dzisiejszej „drobnej" elektronice bardzo ważna jest precyzja i nie da rady lutować starą, poczciwą transformatorówką. Ja osobiście wyposażyłem swój warsztat w kombajn firmy Aoyue model Int2702A+. Dzięki temu mam pod ręką lutownicę, odsysacz pneumatyczny i hotair. Być może ktoś nie widzi zastosowania hot air'a przy konstrukcjach robotów, ale prawda jest inna. Wystarczy, że natrafimy na procesor w obudowie TQFP czy QFN i okaże się, że bez tego typu urządzeń prawie niemożliwa okazuje się praca. Oczywiście okaże się także, że hot air doskonale nadaje się do kształtowania mas plastycznych, lub gięcia tworzyw sztucznych. Cążki sugeruję zakupić dobre, gdyż inaczej odbije się to na pourywanych padach i niepotrzebnych nerwach. Przede wszystkim kupując tego typu narzędzia należy sprawdzić czy „szczęki" schodzą się precyzyjnie oraz czy nie ma luzów poprzecznych. Lutowie... jeżeli ktoś woli, może zastosować cynę zwykłą, ale ja ostatnio przeszedłem na bezołowiową i póki co muszę stwierdzić, że bardzo dobrze się nią lutuje. Podaję swoje praktyczne spostrzeżenia dotyczące wyposażenia, gdyż mogą okazać się pomocne dla osób zaczynających swoją przygodę z robotyką.

Wracając do tematu: gdy już polutujemy zworki, osadzamy dalsze elementy, takie jak rezystory, dławik, kondensatory, podstawki, gniazda i stabilizator napięcia. Oczywiście zbędne wystające końcówki należy usunąć po lutowaniu, by nie doprowadziły do zwarcia. Zostało nam do wykonania bardzo ważne połączenie. Otóż w zamyśle nasz robot będzie posiadał elektronikę w postaci tzw. „kanapki". Do tego jej realizacji wykorzystamy tzw. złącza szpilkowe. Na **fotografii 14** widać, jak należy wykonać takie połączenie, a raczej jak przylutować gniazdo od strony druku.

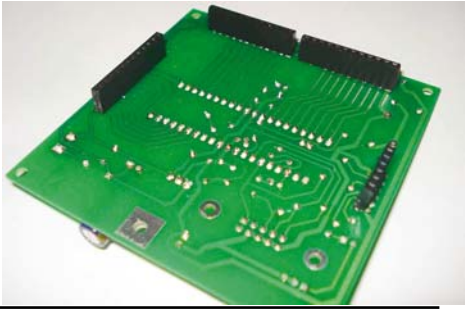
Niby nic trudnego, ale… Jak się okazuje, gniazdo należy unieść około 1 mm nad płytkę i lutować cienkim grotem od strony druku. Dla ułatwienia można lekko pochylić gniazdo a po przylutowaniu delikatnie wyprostować.

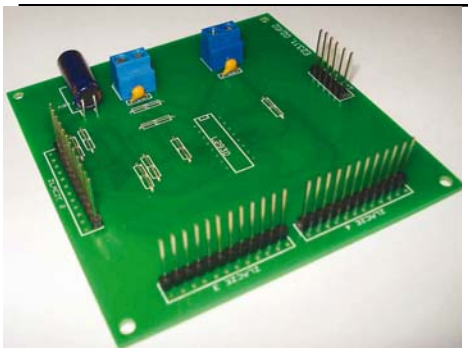
Fot. 12



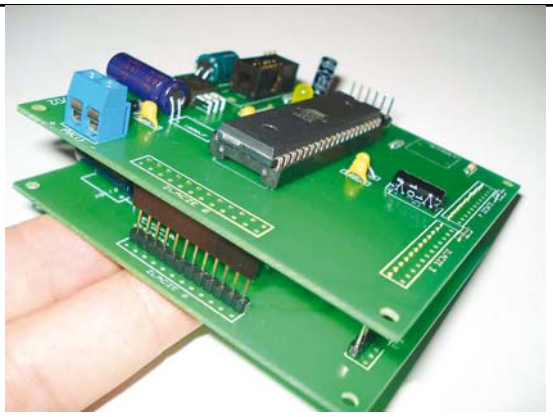
Fot. 13

Fot. 14





Fot. 15



Fot. 16

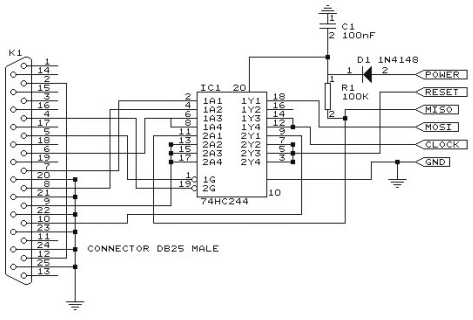
Gniazdo, jak widać, należy zamontować na płytce „mózgu" naszego robota, gdyż będzie to płytka znajdująca się na samej górze ze względu na złącze ISP do programowania procesora.

UWAGA! Na tym etapie nie wkładamy procesora do podstawki.

Gdy już uporamy się z montażem gniazd i elementów płytki mózgu, trzeba polutować płytkę sterownika silników. W tym wypadku montujemy na końcu zamiast gniazd złącza szpilkowe, i to normalnie, od strony elementów, jak pokazuje **fotografia 15**. Tutaj jak widać, jeszcze nie został wlutowany układ scalony L293D. Proszę zwrócić uwagę, iż w płytce dla każdego złącza przewidziane są dwa rzędy otworów, a montowane szpilki tworzą tylko jeden rząd. Co istotne, wszystkie złącza wlutowane są tu od wewnętrznej strony płytki, czyli w rządkach otworów, bliższych środka płytki. A to z tego powodu, że w zewnętrzne otworki, bliższe krawędziom, będą wmontowane od strony ścieżek, podobnie jak w płytce „mózgu", przejścia do kolejnej, trzeciej płytki elektroniki robota. Będzie to moduł komunikacji z otoczeniem, czyli wzmacniacze czujników optycznych.

Jeżeli wszystko zostało poprawnie złożone i polutowane, możemy przystąpić do sprawdzenia poprawności działania modułu „mózgu" tj. podłączamy zasilacz około 7...12V w miejsce wejścia zasilania (aku) i sprawdzamy, czy nie ma zwarcia bądź czy nie grzeje się stabilizator. Jeżeli wszystko jest O.K., to sprawdzamy napięcie na nóżkach GND i VCC procesora. Powinno wynosić około 5V. Jeżeli jest właściwe, to odłączamy zasilanie i odczekujemy czas potrzebny na rozładowanie kondensatorów filtrujących.

Rys. 5



Następnym etapem jest włożenie procesora do gniazda. Po tym zabiegu mamy gotowy „mózg" oraz sterownik silników. Mój model zmontowany na płytkach próbnych pokazany jest na **fotografii 16**.

Daje się zauważyć brak na płytce mózgu wlutowanego przycisku START, a to dlatego, że przycisk ten może być wlutowany także na przewodzie. Będzie to zależało, drogi Czytelniku od tego, jaką zbudujesz konstrukcję mechaniczną swojego robota. Ale o tym w dalszych częściach serii.

Projekty płytek już są umieszczone w Elportalu (*www.elportal.pl/robot)*. Możesz je już wykonać. W chwili pisania artykułu nie wiem, kiedy gotowe płytki pojawią się w ofercie AVT – sprawdź aktualną ofertę na stronie *www.sklep.avt.pl.*

W każdym razie przygotuj się na całość prac. Skompletuj płytki i elementy. Możesz je zmontować. Jeśli masz odpowiednią wiedzę, możesz też je zaprogramować. Ale dla wielu programowanie to czarna magia, więc tej kwestii poświęcimy wiele uwagi.

Zaprogramowanie procesora

Wbrew pozorom, zaprogramowanie mikroprocesora wcale nie jest trudne i na pewno wspólnie poradzimy sobie z tym zadaniem. Jednak do dalszych prac związanych z tym modułem, niezbędne będzie zainstalowanie w komputerze dwóch pakietów oprogramowania: **AVRStudio** oraz pakietu **WinAVR**. Oto linki do tych programów:

www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725

http://winavr.sourceforge.net/

Niezbędne będzie również zaopatrzenie się w **programator zgodny ze standardem STK200/300** lub podobny, który umożliwi programowanie naszego procesora poprzez złącze ISP. Dostępne są gotowe programatory, np.:

www.sklep.avt.pl/p/pl/485091/avtprog1+programator+avr+ispusb+stk500v2.html

Ale zamiast kupować, można samodzielnie zrobić, np. według wzoru ze strony *www.lancos.com/e2p/betterSTK200-mini.gif* lub na przykład według schematu z **rysunku 5**.

Pasujących do naszych celów programatorów jest mnóstwo. W razie wątpliwości i kłopotów będzie można pisać na mój e-mail, postaram się pomóc. Prawdopodobnie jednak poświęcimy cały odcinek tej kwestii.

I co dalej?

Po lekturze tego odcinka przygotuj się do realizacji dwóch opisanych tu płytek. W następnej części omówimy dalsze szczegóły, pokażę kolejne fotografie. Podam także informacje dotyczące budowy bardzo ważnego modułu czujników. Będzie to następny element naszej kanapki. Podam także wskazówki dotyczące instalacji pakietu programów podanych w tym odcinku. Następnie zaczniemy pisać szablon programu oraz zobaczymy pierwsze efekty poprawnej pracy dotychczas skonstruowanej części robota.

Będę wdzięczny za wszelkie sugestie dotyczące dalszego rozwoju projektu. Także w razie pytań będzie można pisać na jeden z moich e-maili.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

Wykaz elementów

Wykaz elementów płytki „mózgu":

R1 . . . . . 4,7kΩ
R2 . . . . . 1kΩ
R3 . . . . . 390Ω
C1,C4,C5. . . . . 100nF/63V
C2 . . . . . 100µF/16V
C3 . . . . . 470µF/6,3V
C6,C7 . . . . . 100uF/6,3V
D1 . . . . . 1N4007
D2 . . . . . LED 5mm
U1 . . . . . LM1117-5V – stabilizator 5V LowDrop w obudowie TO220
U2 . . . . . ATmega16 obudowa DIP
SW1 . . . . . przycisk START – dowolny przycisk NO (normalnie otwarty)
Srebrzanka (do wykonania zworek)
Podstawka 40-nóżkowa
Gniazda na złącza „szpilkowe", można kupić długie i dociąć na odpowiednią liczbę styków
Złącze ISP10 pin – dowolne, może być w obudowie zabezpieczającej przed odwrotnym odłączeniem

Wykaz elementów sterownika silników:

C1 . . . . . 1000µF/16V
C2,C3 . . . . . 100nF/63V
U1 . . . . . L293D
S1,S2 . . . . . złącza silników, dowolne
Srebrzanka na zworki (0,8mm średnicy)
Złącza szpilkowe długie (im dłuższe, tym lepiej), ale można również płytki łączyć przewodami

Komplet podzespołów z płytką każdego modułu będzie dostępny w sieci handlowej AVT

Robot dla każdego, czyli także dla Ciebie część 2

W tym odcinku zaprezentuję informacje, dotyczące oprogramowania oraz tego, by nasz robot ożył. Pokażę, jak należy prawidłowo zainstalować oprogramowanie oraz pomogę postawić pierwsze kroki w pisaniu oprogramowania. Podam kilka praktycznych wskazówek, co i gdzie znaleźć w Internecie. Po zakończeniu tej części nasz „mózg" ożyje i będzie można w rzeczywistości poczuć, że coś się dzieje.

Instalacja WinAVR

Nasza przygoda zaczyna się od instalowania darmowego pakietu o nazwie WinAVR. A cóż to takiego? Osoby obeznane w temacie pewnie już się podśmiewają... ale niniejszy cykl ma pomóc dosłownie każdemu, kto zechce uruchomić własnego robota. Toteż proszę o wyrozumiałość fachowców w tej dziedzinie. Otóż pakiet WinAVR to darmowe środowisko programistyczne z wbudowanym kompilatorem. Kompilator to taki program, który tłumaczy zrozumiały dla nas język na język zrozumiały dla mikroprocesora (mikrokontrolera).

Nasz kurs będzie bazował na języku C. Dlaczego? Otóż język ten daje akurat znacznie większe możliwości od pozostałych znanych języków programowania, przy zachowaniu w miarę prostego sposobu pisania kodu. Jeżeli ktoś miał styczność z językiem C dla komputerów, to z radością donoszę, że ten dla mikrokontrolerów jest bardzo podobny. Jeśli nie, to też spróbujemy sobie jakoś poradzić. Pakiet można pobrać za darmo ze strony: **http://sourceforge.net/projects/winavr/files/WinAVR/20081205/WinAVR-20081205-install.exe/download** Następnie należy rozpakować na dysk komputera i uruchomić instalator. Wybieramy polską wersję językową oraz klikamy przycisk OK – instalator dalej nas poprowadzi. Ważną rzeczą jest, by pozostawić domyślny katalog, w którym program się zainstaluje. Dlaczego? Ponieważ środowisko WinAVR ma swoje wady. Jedną z większych jest to, że w kolejnych wersjach, które się pojawiają, następują znaczące zmiany i bardzo często nie da się uruchomić programu pisanego w poprzedniej wersji. Toteż podam sztuczkę, dzięki której będzie można troszkę oszukać to oprogramowanie. Po instalacji programu należy zrestartować komputer.

Opis dotyczy środowiska Windows, bo akurat takiego używam osobiście. Z tego, co mi wiadomo, pakiet ten występuje także dla innych systemów operacyjnych. Ja osobiście używam Windowsa XP i na tym systemie poprowadzę całą prezentację oraz instruktaż. W razie kłopotów zawsze możesz, drogi Czytelniku, pisać do mnie na e-maila.

Instalacja AVRStudio

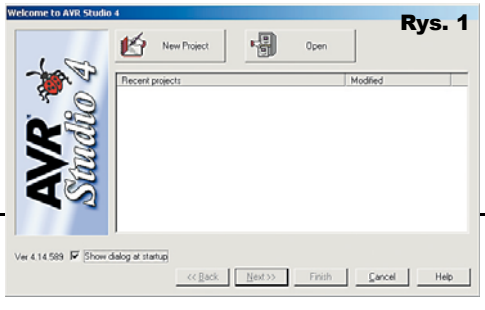
Następnym krokiem jest zainstalowanie środowiska AVRStudio. Jest to pakiet, który należy pobrać ze strony producenta:
www.atmel.com/dyn/products/tools_card_mcu.asp?tool_id=2725
Wszystko jest oczywiście darmowe, ale wymaga troszeczkę zachodu: producent zastrzega sobie, że trzeba się zarejestrować u niego na stronie, by ten pakiet pobrać. Można oczywiście pracować w samym pakiecie WinAVR, ale początkująca osoba „nadzieje się" na konieczność tworzenia pliku „*make*", a to wbrew pozorom nie jest takie łatwe. Dlatego proponuję też zainstalowanie AVRStudio.

Gdy przebrniesz przez proces rejestracji i pobierania wersji instalacyjnej AVRStudio, należy ją odpowiednio zainstalować, by oba pakiety zadziałały prawidłowo.

Zaczynamy instalację AVRStudio – rozpakuj plik lub/i uruchom instalatora. Tak jak w poprzednim pakiecie, radzę zostawić domyślny katalog instalacji. Możesz mi wierzyć, znacznie ułatwia życie pozostawienie obu pakietów na domyślnej instalacji.
Uwaga! Podczas instalacji ważne jest zaznaczenie okienka w ramce „Install/upgrade Jungo USB Driver". Jeżeli program nie zrobi tego automatycznie, zaznacz tę opcję ręcznie. Po dokończeniu instalacji należy wykonać restart komputera i po załadowaniu systemu Windows uruchomić program AVRStudio.

Praca w AVRStudio

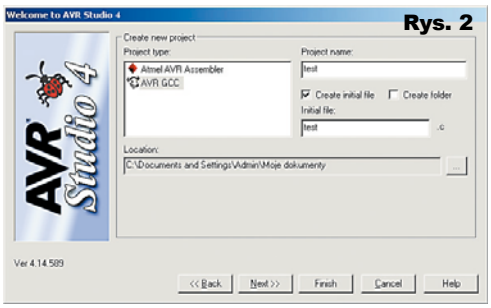
Jeżeli wszystko zrobiłeś prawidłowo i nie było kłopotów podczas instalacji, program powinien się uruchomić i centralnie powinno być widoczne okno pokazane na **rysunku 1**. Teraz przystępujemy do sprawdzenia poprawności działania naszego pakietu oprogramowania, dzięki któremu będziesz mógł sprawić, by Twój robot ożył. Zamierzam pokazać Ci, jak należy pisać proste programy w tym środowisku, ale to od Ciebie będzie zależało, czy będziesz dalej rozwijał swoje umiejętności.



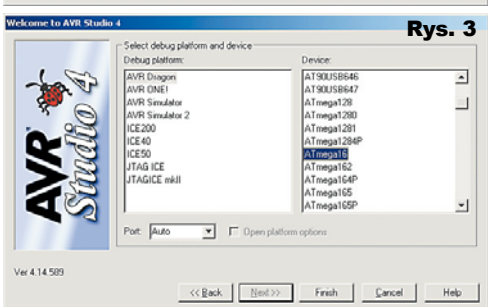
Rys. 1

W widocznym oknie klikamy myszką na duży przycisk „*New Project*". Teraz pokaże się okno widoczne na **rysunku 2**. Co do tego fragmentu, to mam dla Ciebie dwie rady: pierwsza – ***zaznacz od razu (niezaznaczoną domyślnie) opcję „Create folder"***. Uwierz mi, znacznie zmniejsza to bałagan na dysku twardym i upraszcza pracę. Druga rada: wybierz (podświetl) **AVR GCC** w dużym oknie „*Project type*". Teraz jeszcze w okienku tekstowym po prawej stronie zatytułowanym „*Project name*" podaj nazwę projektu. Ja dla przykładu wpisałem „***test***". Następnie naciśnij przycisk „*Next*" a zobaczysz kolejne okno widoczne na **rysunku 3**. Tutaj do wyboru masz całą gamę procesorów, ale nas interesuje taki procesor, jaki będzie znajdował się w przyszłym robocie. Ja osobiście wybrałem ATmega16, bo opisana miesiąc temu płytka „mózgu" zawiera właśnie taki procesor. Teraz klikamy przycisk „Finish" i zobaczymy to, co na **rysunku 4**.

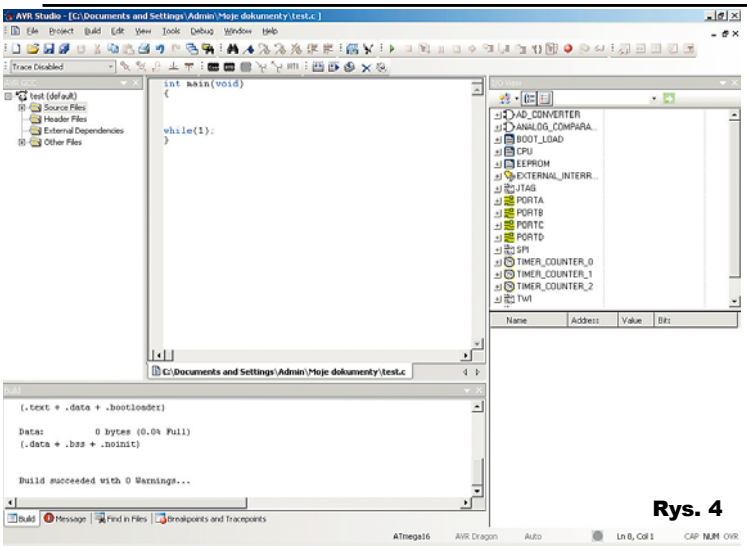
Tak, Drogi Czytelniku – to jest Twój i mój pulpit roboczy. Zawiera on kilka okienek rozmieszczonych wokół jednego większego. To największe okno będzie służyć do pisania i modyfikowania programu. Po prawej stronie jest okno z informacjami o procesorze. Po lewej okno z informacjami o plikach i katalogach, użytych w projekcie. A pod oknem znajduje się *konsola raportów* – to jest moja nazwa. W każdym razie bardzo istotne są informacje, które wyświetlane są właśnie w tym oknie. Przystępujemy do przetestowania pakietu. W oknie edycyjnym (tym największym) należy wklepać taki oto tekst:



Rys. 2



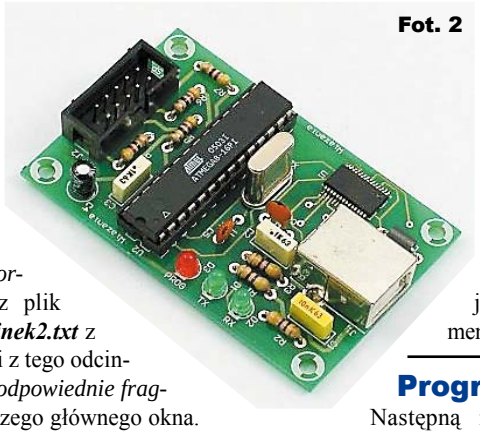
Rys. 3



Rys. 4

```
int main(void)
{
while(1);
}
```

Jeżeli nie pisałeś nigdy programów w języku C, to proszę przepisz to dokładnie. Dla ułatwienia, w Elportalu (*www.elportal.pl/robot*) znajdziesz plik tekstowy o nazwie ***odcinek2.txt*** z wszystkimi programami z tego odcinka. Możesz skopiować *odpowiednie fragmenty tego pliku* do naszego głównego okna.



Fot. 2

Po wpisaniu tego prościutkiego programu, naciśnij proszę przycisk F7 na klawiaturze swojego komputera – zostanie zrealizowane skompilowanie projektu, czyli przetłumaczenie tego, co napisaliśmy w C, na tzw. kod maszynowy, zrozumiały dla procesora (mikrokontrolera).

Jeżeli wszystko podczas instalacji przebiegło pomyślnie, efektem końcowym jest pojawienie się komunikatu (w konsoli raportów) o treści „*Build succeeded with 0 warnings...* ”. Ten komunikat wskazuje, że programy są zainstalowane prawidłowo oraz że połączenie AVRStudio i WinAVR działa prawidłowo. No i w tym miejscu mogę Ci też pogratulować pierwszego działającego programu!

Gratuluję więc!

W obecnej chwili ten program da niewiele efektów wizualnych, gdyż jego działanie polega na wykonywaniu tzw. pętli nieskończonej.

Dla zaawansowanych to nic nowego. Ale niech wszyscy, którzy nie znają języka C, nie popadają w panikę. Nie trzeba wiedzieć wszystkiego. W praktyce zupełnie wystarczające może się okazać korzystanie z gotowych programów i ewentualna ich modyfikacja na własne potrzeby. A to naprawdę nie jest trudne.

A zanim zaczniemy nasze poważne ćwiczenia, proponuję, żeby mało zaawansowani poszukali w Internecie gotowych programów, napisanych w C dla procesorów AVR, najlepiej dla naszego ATmega16.

Niech to będzie zadanie domowe!

Fot. 1

Nie żałujcie na to czasu. Nawet jeżeli niewiele zrozumiecie, nie przestraszcie się. Chodzi o to, żeby nie bać się języka C i programowania. Ważną informacją dla osób początkujących jest to, by zapoznały się ze sposobem pisania programu w języku C. Jak ja zaczynałem, zawsze gubiłem się w średnikach. A to jest bardzo ważny element tego języka.

Programatory

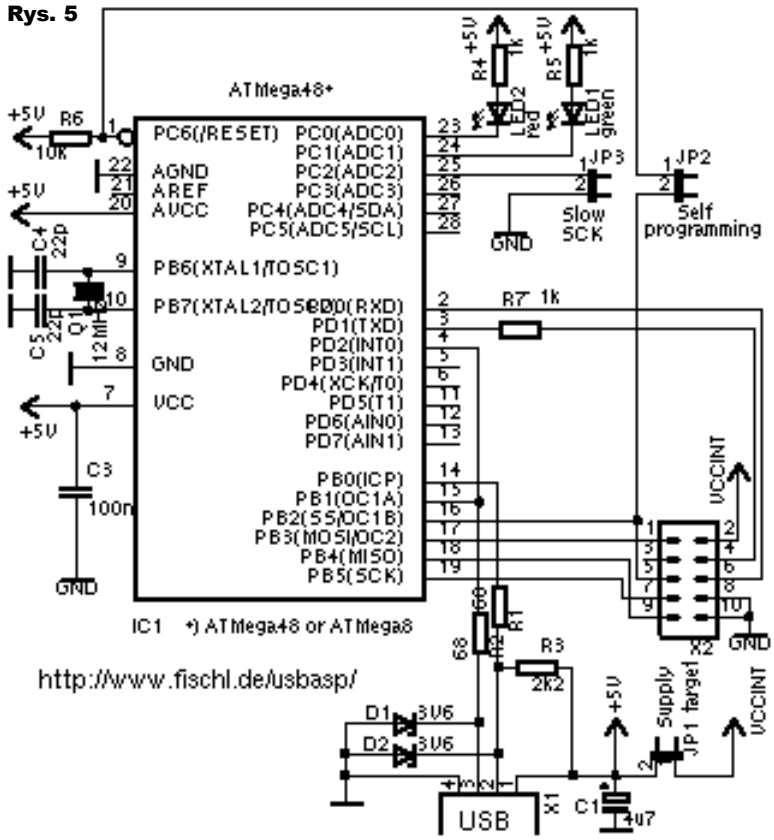
Następną rzeczą, jaką należy wykonać, jest zaopatrzenie się w programator zgodny z STK500. Możemy z powodzeniem użyć bardzo prostego w budowie programatora zgodnego ze standardem STK200/300 (wspomniałem o nim w poprzedniej części kursu). Jest on jednak kłopotliwy z tego względu, że większość nowych komputerów nie ma portu równoległego. Natomiast programator STK 500 pracuje na złączu USB. W związku z tym postanowiłem przeprowadzić rozeznanie, co można kupić oraz co można samemu zbudować. I tutaj mogę przedstawić kilka opcji, w zależności od tego, jakim sprzętem dysponujesz. Dlaczego? Otóż ***wykonanie samemu własnego programatora może okazać się niemożliwe, jeżeli nie posiadasz żadnego sprzętu do zaprogramowania mikrokontrolera***. Sugeruję więc byś kupił gotowy egzemplarz stk500 lub odpowiednika. Z tych, które znalazłem na rynku, mogę zaproponować Ci np. ten dostępny w sieci handlowej AVT: **http://sklep.avt.pl/p/pl/485091/avtprog1+programator+avr+ispusb+stk500v2.html**. Tę solidną konstrukcję możesz zobaczyć na **fotografii 1**.

Następnym programatorem dostępnym w sprzedaży może być programator zawarty w zestawie do montażu – AVT-5125. Dostępny jest między innymi pod tym adresem: **http://sklep.avt.com.pl/p/pl/482096/programator+usb+dla+avr+-+zestaw+do+samodzielnego+montazu.html**. Jego wygląd przedstawia **fotografia 2**.

Cała masa gotowych programatorów dostępna jest także na Allegro. Wystarczy wpisać w wyszukiwarce symbol poszukiwanego programatora, czyli STK500 lub STK500v2. Prawda jest taka, że większość jest budowana na podstawie kilku sprawdzonych projektów, które także są dostępne w Internecie. Ale ponieważ jest to artykuł dla elektroników, więc i my pokusimy się o zbudowanie własnego programatora. Nawet jeżeli nie masz jak zaprogramować mikrokontrolera, zapewne znajdzie się ktoś, kto ten pierwszy pomoże Ci zaprogramować.

Moje propozycje co do programatorów do samodzielnego wykonania zawierają się w kilku

Rys. 5



przykładach. Ale według mnie najlepszą konstrukcją z nich jest ten, prezentowany na stronie: **www.fischl.de/usbasp/**
Jest to całkowicie darmowy projekt programatora zgodnego ze standardem STK500. Konstrukcja jest tak prosta, że każdy może ją wykonać we własnym zakresie. Na **rysunku 5** prezentuję schemat tego urządzenia. Jak widać, w zasadzie programator składa się z jednego mikrokontrolera i kilkunastu elementów rozmieszczonych wokół niego. Nie ma tu wyszukanych i drogich elementów. Programator działa wyśmienicie i z czystym sumieniem mogę go polecić. **Fotografia 3** przedstawia wykonany przeze mnie egzemplarz. Jest to 100% kopia jednej z propozycji wykonania, której fotografia znajduje się na podanej przeze mnie stronie WWW.

Dla mnie największym kłopotem okazało się zdobycie gniazdka kątowego. Tu z pomocą przyszła oferta firmy TME. Wykonanie tego programatora omówię w szczegółach w następnym odcinku cyklu. Aczkolwiek, jeżeli ktoś chce już teraz go zbudować, to zapraszam na podaną stronę WWW – jest tam cała dokumentacja potrzebna do jego wykonania. Również na tej stronie znajduje się kilka linków do innych wykonań tego samego projektu oraz do stron poświęconych programowej obsłudze standardu USB. Bo muszę tu podkreślić genialność konstrukcji tego typu programatorów. Żaden z procesorów (mikrokontrolerów) rodziny ATmega8, 48, 88 itp. nie ma sprzętowej obsługi standardu USB. Ale okazuje się, że można z powodzeniem zaimplementować ją programowo. Polecam zainteresowanym zapoznanie się z tym aspektem użycia procesorów ATmega, bo wcześniej czy później zapewne także i Ty staniesz przed koniecznością zrobienia własnego urządzenia bazującego na USB.

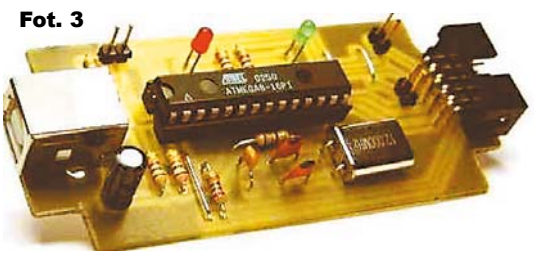
Tyle tytułem wstępu do programowania.

Program

Czas zająć się programem, a dokładniej, postaram się pomóc w zrozumieniu podstaw programowania języka C w środowisku WinAVR połączonym z AVRStudio. Nie będzie to rozbudowany kurs, bo takowych jest sporo w sieci, np. jeden z najlepiej opisywanych znajduje się pod tym adresem:
www.kursc.dioda.com.pl/

Do tego jest cała masa książek na ten temat. My zajmiemy się tylko najbardziej podstawowymi sprawami. Najważniejszą rzeczą, jaką należy pamiętać to to, że w języku C procedury bądź fragmenty podprocedur należy umieszczać w nawiasach {}. Do tego drugą ważną sprawą jest to, że linijkę kodu zawsze (no, prawie zawsze) kończymy znakiem średnika **;**. Jeżeli tego nie będziesz przestrzegać, kompilator automatycznie zasygnalizuje błąd.

Fot. 3



Główną pętlą (procedurą) programu pisanego w tym środowisku jest „main". W kodzie programu wygląda to następująco:

```
int main(void)
{
//… tu reszta kodu
}
```

Tutaj jak widać na zakończeniu „}" nie ma znaku średnika – i tak właśnie ma być.
Jest to pierwsza procedura, jaką wykonuje mikrokontroler – w niej należy umieszczać wszystko, co chcesz zrobić (prawie wszystko, bo przerwania wykonujemy inaczej – ale na ten temat w dalszej części). Aby procesor wykonał jakiekolwiek działanie, należy w klamerkach „main" wpisać, co chcemy, by zrobił, np.:

```
int main(void)
{
while(1);
}
```

Ta procedura nie robi w zasadzie niczego oprócz wprowadzenia procesora (mikrokontrolera) w stan nieskończonej pętli.

Bowiem polecenie *while* oznacza w skrócie „dopóki", a w nawiasie podany jest warunek. Czyli nasze polecenie wykonuje się, dopóki warunek w nawiasie jest większy od „0". A dokładniej, dopóki jest prawdziwy – „TRUE", a w języku C wartość „FALSE" to po prostu „0". Zauważyłeś może, że w pierwszym fragmencie użyłem zwrotu:
„ // … tu reszta kodu"
– wyjaśniam, że w języku C tym podwójnym ukośnikiem „//" oznaczamy komentarz, czyli coś, co jest pomijane przy kompilowaniu programu. Można komentować jedną linię kodu, stawiając na początku „//". Można też komentować większy fragment, używając znaczników początku i końca komentarza: „**/***" oraz „***/**".

I tak oto przeskoczyliśmy nasz pierwszy program testowy, napisany w świeżo zainstalowanym środowisku AVRStudio + WinAVR.

Analiza programu

Bardzo ważnymi elementami procesora są „REJESTRY". W tej części kursu nie będziemy wnikać, do czego wszystkie służą. Zajmiemy się tylko kilkoma najważniejszymi, które pozwolą nam cokolwiek zrobić z procesorem. Dlatego sugeruję, byś teraz wpisał do naszego okna program z **listingu 1**.

Listing 1

```
#define F_CPU 8000000UL
#include „stdio.h"
#include „stdlib.h"
#include <avr/io.h>
#include <util\delay.h>
#include <avr/interrupt.h>
#define cbi(add,bit) ((add) &= ~(1 << bit));
#define sbi(add,bit) ((add) |= (1 << bit));
#define  Led  PORTB  //Do tego pinu podlaczona jest dioda LED2
int i;
int licz;
void migaj_led(int licz)
{
while(i<=licz){
cbi(Led,2);
_delay_ms(20);
sbi(Led,2);
_delay_ms(60);
++i;
}
}

int main(void)
{
sbi(DDRB, 2);
do
{
i=0;
migaj_led(2);
_delay_ms(800);
}
while(1);
}
```

Po wpisaniu naciśnij przycisk „F7" na klawiaturze – skompiluj program. Jeżeli wszystko jest dobrze zainstalowane oraz nie popełniłeś błędu, to powinieneś zobaczyć komunikat o tym, że nie ma błędów (*errors*) ani ostrzeżeń (*warnings*). Ostrzeżenia są dużo mniej znaczącymi informacjami, ale zawsze sugeruję zerknąć, o co chodzi. Błąd przy kompilacji uniemożliwi stworzenie pliku *.hex, który jest wynikiem kompilacji. Plik *.hex jest to zawartość, którą następnie wgramy do mikrokontrolera za pomocą naszego programatora.

Co teraz zrobi nasz program? Otóż program ten zrobi nam miłą niespodziankę, jaką będzie migająca równomiernie dioda LED na opisanej w EdW 2/2010 płytce „mózgu" naszego robota. Wreszcie zobaczysz, że płytka ta rzeczywiście działa. Wyjaśnię teraz, co nasz program robi. A później opiszę, jak sprawić, by znalazł się w pamięci mikrokontrolera. Zaczynamy krótką analizę tego, co widzimy w programie.
#include (z języka angielskiego) oznacza – *zawiera*. Polecenie to służy do tego, by poinformować kompilator, by użył jakiegoś innego pliku, który spełnia określone zadania.
#define F_CPU 8000000UL – bardzo ważny parametr – wskazuje kompilatorowi, jaka jest częstotliwość taktowania procesora (jak szybko działa).
Upraszczając bardzo zawartość:
#include „stdio.h" oraz **#include „stdlib.h"** zawierają opisy większości „gotowych" komend, które będziemy stosować w następnych częściach cyklu.

Bardzo ważna biblioteka
#include <avr/io.h> zawiera informacje o tzw. rejestrach mikrokontrolera. Zawarte są w niej definicje wszystkich ważnych elementów naszego procesora, żebyśmy przy pisaniu programów mogli posługiwać się zrozumiałymi dla nas oznaczeniami, takimi jak PORDx, PIN, DDRx, itd. W następnych częściach cyklu postaram się wytłumaczyć, co jak oraz do czego służy. Na teraz staram się wytłumaczyć w ogólnym zarysie, co robi nasz program.

Polecenie **#include <avr/delay.h>** oznacza, że kompilator ma użyć pliku z poleceniami opóźnień, które są zawarte właśnie w tym pliku **delay.h**. Zawsze możesz napisać własne procedury, ale sugeruję, by na początku używać gotowców.

Następna ważna biblioteka, która będzie nas interesować w późniejszym czasie, to:

#include <avr/interrupt.h> – zawiera ona informacje o obsłudze przerwań. Dla niewtajemniczonych, mówiąc prostym językiem, są to specjalne funkcje procesora (mikrokontrolera), które pozornie mogą się wykonywać jednocześnie z działaniem reszty programu. Kiedy będziemy używali przerwań? W następnych częściach kursu... Przerwania będą nam potrzebne do obsługi między innymi czujników odbiciowych.

Polecenia **#define …** pozwalają prościej zapisywać pewne działania/rozkazy. Zamiast pisać za każdym razem takie oto polecenie: **((add) &= ~(1 << bit))**, użyjemy jego skrótu, stworzonego właśnie za pomocą **#define**, czyli używać będziemy w programie polecenia **cbi(add,bit)**. Uwierz mi, że znacznie ułatwia to pisanie programu. Tłumacząc w skrócie, polecenie **cbi** powoduje wyzerowanie określonego bitu, a polecenie **sbi** powoduje ustawienie określonego bitu (przyjmuje odpowiednio wartość 0 i 1.

#define cbi(add,bit) ((add) &= ~(1 << bit));

#define sbi(add,bit) ((add) |= (1 << bit));

Prócz na pozór tak zagmatwanych przyporządkowań, możemy stosować także prostsze, takie jak to:

#define Led PORTB – nie musimy pamiętać, gdzie jest podłączona dioda LED, wystarczy pamiętać, że mamy użyć słowa „Led". Proszę też pamiętać, że ***w języku C ważna jest wielkość liter***. Zawsze na początku myliłem ***Led*** z ***led***, a to nie to samo.

Myślę, że już czujesz z grubsza, o co chodzi z poleceniami **#include** i **#define**. Zajmijmy się zatem dalszą częścią programu. Są to definicje zmiennych tzw. globalnych. W języku C mamy dwa podstawowe rodzaje zmiennych. Są to zmienne „lokalne" oraz zmienne „globalne". Podstawowa różnica polega na tym, że zmienna globalna dostępna jest prawie w każdym punkcie programu, a zmienne lokalne dostępne są w obrębie danej procedury (podprogramu). Starajcie się nie używać zbyt dużej liczby zmiennych globalnych, gdyż zajmują one głównie pamięć RAM mikrokontrolera – a tej zawsze za mało.

W naszym programie użyjemy dwóch zmiennych: **int i;** oraz **int licz;**. Słowo **int** oznacza, w skrócie, zakres wartości jakie może przyjąć (integer). I tak w języku C dostępne mamy takie oto podstawowe „zakresy" (typy zmiennych):

char – zmienna przechowuje znaki (litery, cyfry, znaki interpunkcyjne). Za pomocą tego typu zmiennej można także przechowywać niewielkie liczby,

int – zmienna służy do przechowywania liczb całkowitych,

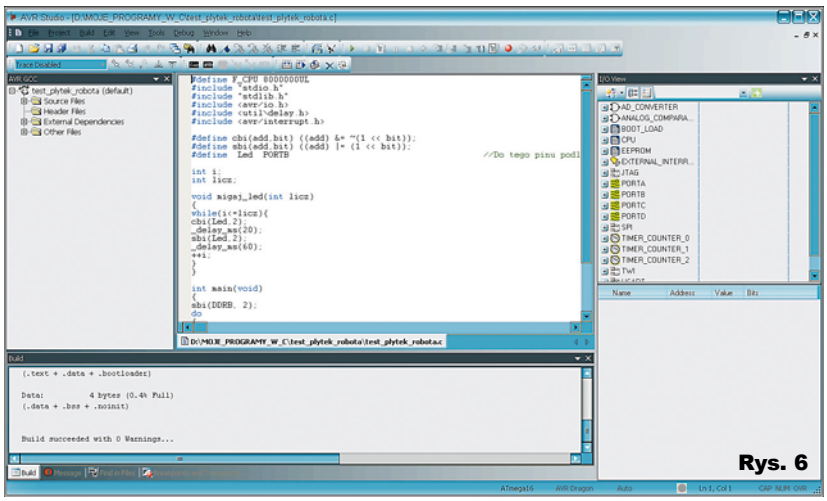
float – zmienna przechowuje liczby rzeczywiste (zmiennoprzecinkowe – do 7 cyfr po przecinku),

double – zmienna przechowuje liczby rzeczywiste podobnie jak powyższe (ale do 15 miejsc po przecinku).

Język C pozwala także na użycie pewnych „kwalifikatorów" przed typem zmiennej:

signed – zmienna może przechowywać wartości dodatnie i ujemne (posiada znak +/-),

unsigned – zmienna może przechowywać tylko wartości dodatnie,


Rys. 6

short – zmienna jest typu krótkiego – wpływa na długość zajmowanej pamięci (a więc również na zakres zmiennej),

long – zmienna jest typu długiego.

Istnieje także możliwość określenia „klasy" zmiennej. Należy jednak podkreślić, że TYP ZMIENNEJ (**char**, **int**, itp.) decyduje o sposobie interpretacji przechowywanych w pamięci bitów, natomiast KLASA ZMIENNEJ decyduje o sposobie przechowywania zmiennej w pamięci. W C występują następujące klasy zmiennych:

const – stała – nie można zmieniać wartości raz nadanej,

static – zmienna w momencie uruchomienia programu otrzymuje stałe miejsce w pamięci. UWAGA: Słowo kluczowe *static* ma inne zastosowanie przy deklaracji funkcji,

volatile – oznacza zmienną, z której mogą korzystać także inne procesy. Oznacza to np., że inny proces może zmieniać wartość tej zmiennej, przez co przed każdym jej użyciem musi zostać odczytana na nowo (jest to bardzo ważna klasa zmiennych – dzięki niej możemy poprawnie używać przerwań – o tym w następnej części cyklu).

W celu wyjaśnienia dokładnych różnic oraz pozostałych odmian zmiennych, zapraszam do odwiedzenia: **http://pl.wikibooks.org/wiki/C/Zmienne**. Nie jestem w stanie w ramach tego cyklu wytłumaczyć wszystkiego dokładnie, toteż pozostaniemy tylko przy analizie tego, co nasze programy będą robić, a chętnym będę podawał linki do dokładniejszych informacji. Kurs ten bowiem ma na celu zbudowanie robota, a nie przeprowadzenie wnikliwego kursu języka C. Proszę zatem o wybaczenie, ale skoncentrujmy się dalej na analizie naszego programu, gdzie widnieje taki oto podprogram:

```
void migaj_led(int licz)
{
while(i<=licz){
cbi(Led,2);
_delay_ms(20);
sbi(Led,2);
_delay_ms(60);
++i;
}
}
```

Pamiętasz zapewne zasady pisania „{}" oraz używania *while*. Tutaj masz pokazaną w pełnej krasie procedurę powodującą miganie diody LED, przyłączonej do naszego mikrokontrolera. Wywołanie naszego podprogramu wymaga podania liczby „mignięć", a zdefiniowane jest to w następujący sposób: **void migaj_led(int licz)**. Następnie wykonuje się pętla: **while(i<=licz){}**. Wewnątrz tej pętli znajduje się szereg poleceń: **cbi(Led,2);** powoduje „zwarcie" nóżki procesora do masy, **_delay_ms(20);** powoduje odliczenie 20ms (nieprzypadkowo na początku programu użyliśmy biblioteki *delay.h*), **sbi(Led,2);** powoduje przejście „nóżki" procesora do stanu wysokiego – logicznej 1. Dzięki wcześniejszemu użyciu **#define Led PORTB** wiemy, że używając poleceń *sbi* oraz *cbi*, aktywujemy odpowiednią „nóżkę" procesora. Użycie określenia **cbi(Led,2)** oznacza aktywację („zwarcie" do masy) nóżki określonej jako PORTB bit 2. Uff... trochę to może wydać się skomplikowane, ale wierz mi, że jeżeli poeksperymentujesz, to wszystko stanie się jasne. Dalej jest polecenie **_delay_ms(60);** – nie muszę go tłumaczyć. Ostatnie polecenie to: **++i;** Powoduje ono za każdym razem zwiększenie zmiennej **i** o 1. Innymi słowy, cały podprogram zostanie wykonany „*Licz*" razy, przy czym pamiętajmy, że liczymy od 0. Czyli gdy podamy 3, to dioda mignie 4 razy.

Dalej nasz program zawiera główną pętlę programu, czyli „main" – jak pisałem już wcześniej, mikrokontroler najpierw wykona to, co znajduje się w pętli main. Dzięki temu wszystkie ustawienia niezwiązane z definicjami należy wywoływać w „main". Cóż robi nasza procedura main? Po kolei fragment kodu:

```
int main(void)
{
sbi(DDRB, 2);
```

```
do
{
i=0;
migaj_led(2);
_delay_ms(800);
}
while(1);
}
```

A teraz opis. BARDZO ważne jest pierwsze polecenie: **`sbi(DDRB,2);`** Pamiętasz, jak wspominałem o rejestrach… Rejestr *DDRx* to tzw. rejestr kierunkowy portów I/O mikrokontrolera. Określa on wstępnie, co będzie robić każda „nóżka" układu. Ustawienie danego bitu rejestru DDRx (wpisanie 1 logicznej) powoduje, że dana „nóżka" staje się wyjściem, natomiast wpisanie logicznego „0" powoduje, że dane wyprowadzenie staje się wejściem. Innymi słowy, użycie naszego polecenia powoduje, że PORTB bit 2 staje się wyjściem. Jeżeli nie wpiszemy tego polecenia, procesor nie będzie sterował zewnętrznym wyprowadzeniem – dioda LED nie będzie nam działać.

Pętla do **`{} while(1);`** powoduje, że będzie wykonywać się w nieskończoność to, co jest zawarte w klamerkach {}. Efektem tego będzie: i=0; zmienna **i** przyjmuje wartość 0, migaj_led(2); wywołujemy wcześniej omówioną procedurę migania, _delay_ms(800); odczekujemy 800ms i znów powtarzana jest pętla, począwszy od i = 0;.

Ot i cała filozofia. Jak widzisz, język C jest w istocie prosty. Wymaga tylko zrozumienia pewnych zagadnień i przyzwyczajenia się do specyficznego zapisu. Reszta pójdzie z górki. W razie pytań zawsze można do mnie pisać, postaram się pomóc.

Jak wgrać program do procesora?

Napisany przez nas program należy skompilować i wgrać do procesora. Jak to zrobić? Poniżej zamieszczam skrócony poradnik – co i jak ustawić. W następnej części dotyczącej programowania wyjaśnię, dlaczego pewne rzeczy należy ustawić tak, a nie inaczej. Pierwszą rzeczą, jaką należy wykonać, jest zmontowanie płytki „mózgu" z poprzedniej części kursu. Gdy są one już gotowe, możemy wykorzystać samą płytkę „mózgu" lub spiąć obie razem. Nie spowoduje to żadnych uszkodzeń. Mając programator oraz płytki, należy spiąć wszystko razem. Podłączamy zasilanie do płytki „mózgu" w miejsce oznaczone jako AKU. Wartość napięcia obecnie nie jest krytyczna, jeżeli na płytce modułu znajduje się stabilizator 5V. Jeżeli go nie zastosowałeś, to musisz podpiąć 5V, inaczej uszkodzisz mikrokontroler. Co innego, gdy stabilizator jest na płytce. Wtedy wartość napięcia zasilania, jaką można podłączyć, leży w zakresie 6,5 V–10V (a nawet 12V). Następnie podpinamy programator do komputera, uruchamiamy AVRStudio z napisanym wcześniej programem. Podpinamy programator do płytki. Teraz należy w AVRStudio skompilować program według wcześniejszego opisu (F7). Jeżeli wszystko zadziałało, to OK. Powinieneś zobaczyć okno z komunikatem widocznym na **rysunku 6**. Oznacza to, że nie ma błędów i można plik wgrać do procesora. Teraz wywołujemy okno programatora i wybieramy typ procesora. W moim przypadku jest to ATmega16, co widać na **rysunku 7**. Po wybraniu typu procesora klikamy przycisk „*Read Signature*". W dolnej części ekranu pojawią się jakieś teksty, a pod okienkiem wyboru procesora powinieneś zobaczyć napis „*Signature matches selected device*". Po tej operacji mamy pewność, że z mikrokontrolerem oraz programatorem wszystko jest OK. Następnie przechodzisz do zakładki „*Fuses*" i ustawiasz wszystko tak jak na **rysunku 8**. ***To jest bardzo ważna operacja! NIE POMYL się, bo grozi to uszkodzeniem mikrokontrolera***.

Jeżeli wszystko ustawiłeś tak, jak pokazano na **rysunku 9**, naciskasz przycisk „***Program***". Niekiedy może wyskoczyć komunikat o tym, czy na pewno chcesz wyłączyć JTAG – po prostu zatwierdzasz, że tak. Po wykonaniu tej operacji, w dolnej części okna pojawi się napis „*Leaving programming mode. OK!*". I od tej chwili nasz procesor taktowany jest zegarem wewnętrznym 8MHz. Następną czynnością jest kliknięcie na zakładce „*Program*" i wybranie zawartości „*Flash*". Należy tu podać ścieżkę dostępu do katalogu, w którym zapisałeś na początku swój projekt. Plik *.hex znajduje się w tym katalogu, w podkatalogu: *default* i ma nazwę taką samą, jaką nadałeś, tworząc projekt. Sugeruję również, byś Drogi Czytelniku poustawiał sobie wszystko na tym oknie tak, jak jest to widoczne na rysunku 9.

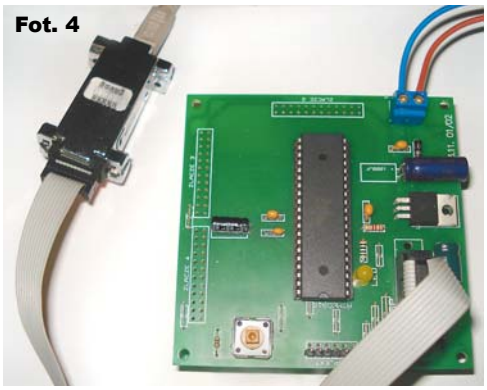
Po wybraniu pliku i naciśnięciu przycisku „*Program*" w sekcji „*Flash*", otrzymasz po chwili komunikat „*Leaving programming mode. OK!*" i od tej chwili „mózg" Twojego robota ożywa.

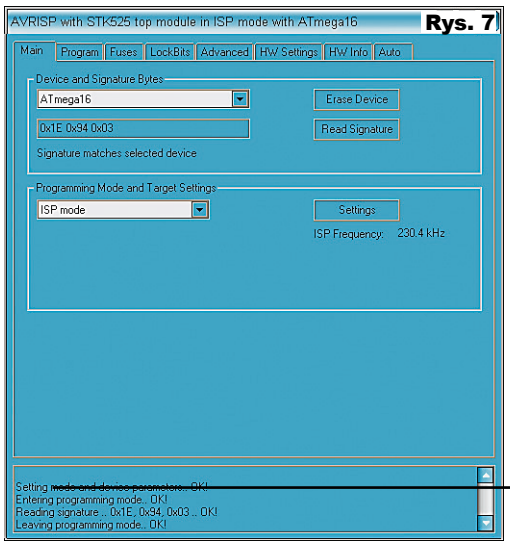
Fotografia 4 pokazuje moją płytkę mózgu i programator podczas opisywanych właśnie operacji.

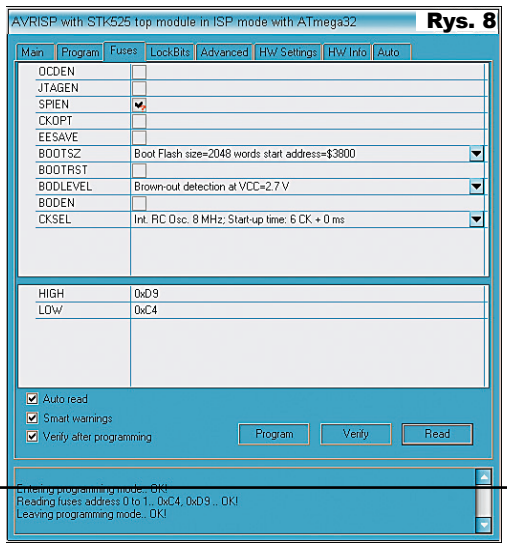
Zachęcam do zabawy z podanym przykładem programu. Poćwicz różne opóźnienia, różne liczby mignięć, samodzielnie zaprojektuj bardziej efektowne sekwencje. Zapoznaj się z linkami podanymi w tej części cyklu. A w razie kłopotów pisz do mnie na adres e-mail.

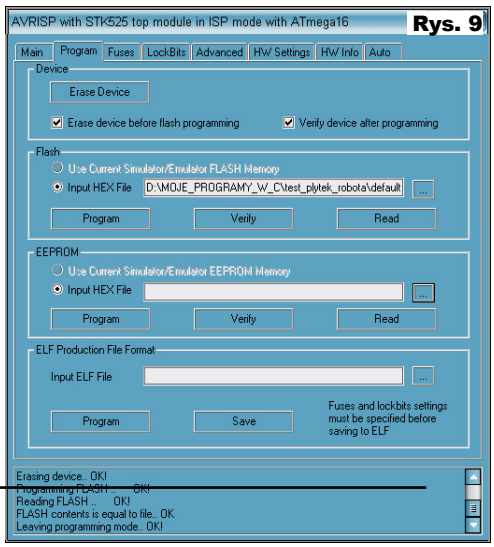
W elektronice najważniejsze jest eksperymentowanie i brak obaw, że może się coś uszkodzić. Oczywiście potrzebny do tego jest zdrowy rozsądek i trochę wyobraźni. Ale jeśli czytasz ten artykuł, tego na pewno Ci nie brakuje. W dalszej części cyklu dowiesz się, jak spowodować, by płytka „mózgu" reagowała na bodźce zewnętrzne, do czego służą przerwania, co to jest PWM, trochę o rejestrach i masę innych ciekawych informacji.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

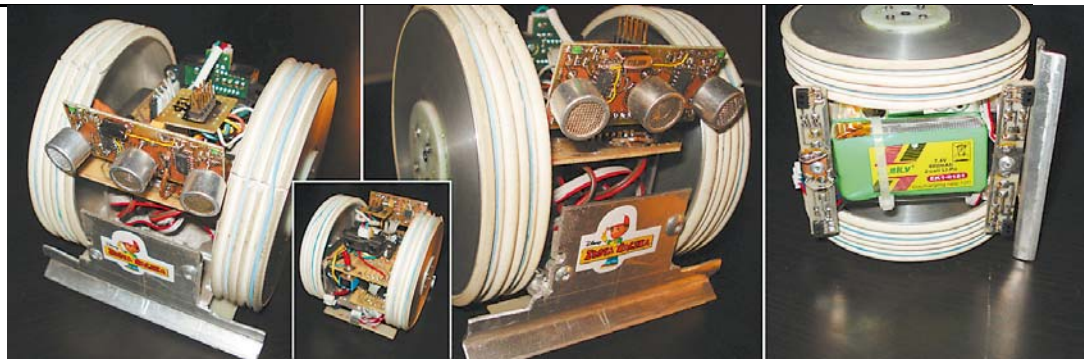

Fot. 4







kit 2935 AVT



Robot mobilny – krok po kroku

część 3

Mam nadzieję, że zgodnie z moim zaleceniem ćwiczyliście różne kombinacje w programie, a także podszkoliliście się trochę w obsłudze AVRStudio. W tym odcinku pójdziemy o krok dalej, ale nie będziemy bez potrzeby powracać do rzeczy omówionych w poprzedniej części artykułu. Dlatego w razie wątpliwości przede wszystkim trzeba powrócić do materiału sprzed miesiąca.

Własny programator na USB

Jak obiecałem w poprzednim odcinku, na początku zajmiemy się uruchomieniem programatora. Programator, który zaprezentuję w tej części artykułu, powstał na bazie projektu, o którym wspomniałem w poprzednim odcinku (**http://www.fischl.de/usbasp/**) – jest on na licencji GNU. Jednakże jest to projekt, który wznowiłem (zaprojektowałem od nowa płytkę i przetestowałem prototyp) specjalnie dla tego cyklu. Na **fotografii 1** widać gotowy programator, **rysunek 1** przedstawia płytkę drukowaną. Została ona zaprojektowana w programie KiCad – plik źródłowy płytki można ściągnąć z Elportalu (jak również pliki PDF). Programator ten jest bardzo prosty do wykonania. Jednakże procesor, który w nim się znajduje, wymaga pierwszego programowania. W zastawie AVT-2935 procesor będzie oczywiście zaprogramowany. Od biedy programowanie można zrealizować za pomocą kilku przewodów i programu pracującego na złączu LPT1 lub poprosić kogoś znajomego by zaprogramował tę pierwszą kostkę. Można również przesłać pocztą do mnie lub podjechać, chętnie wykonam takie programowanie za darmo. Mój e-mail podany jest na końcu artykułu. Niestety nie unikniemy tej pierwszej niedogodności, gdyż programator ten bazuje na programowej emulacji portu USB i dzięki temu, że nie zawiera specjalizowanych układów, takich jak FTDI, jest bardzo prosty i tani.

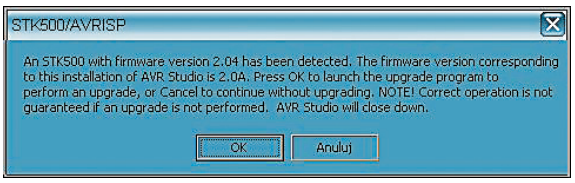
Ma dwie możliwości pracy. Oficjalną, która współpracuje tylko z oprogramowaniem Avrdude oraz mniej oficjalną, która emuluje tryb STK500. Nasz poprzednio instalowany pakiet będzie się co prawda buntował, że jest to stara wersja STK500, ale nie należy się tym przejmować i zawsze klikać ***Nie*** (***Anuluj***) jeżeli chce aktualizować nasz programator. Sytuację taką uwidoczniłem na **rysunku 2**.

Co dalej? Gdy już mamy zaprogramowany programator, należy w Windowsie zainstalować sterowniki do niego. Sterowniki można pobrać z tego adresu: **http://www.obdev.at/downloads/vusb/AVR-Doper.2008-11-27.zip**. Instaluje się je bardzo prosto. Następnie należy podłączyć programator, zaczekać aż Windows zainstaluje urządzenie USB i gotowe. Odpalamy AvrStudio i ładujemy nasz projekt. Mam nadzieję, że w międzyczasie bawiliście się nim trochę :-). Dla pewności proszę skompilować projekt (przycisk F7). Jeżeli wszystko jest OK, to wchodzimy do menu *programowanie*. Wybieramy typ programatora STK500 oraz port przypisany do programatora (można to sprawdzić w menedżerze urządzeń, dopuszczalny zakres portów 1-4). Teraz ładujemy plik *.hex naszego projektu. Podpinamy programator do płytki „mózgu" robota (korzystając ze złącza SPI) i sprawdzamy, czy komputer „widzi" procesor.

Jeżeli tak i jeśli pisze, że parametry są zgodne, przechodzimy na zakładkę z wcześniej wybranym plikiem *.hex i wciskamy *zaprogramuj*. Jeżeli wszystko jest OK, to powinno po chwili wyskoczyć okienko, że programowanie zakończono sukcesem. Dokładnie komunikacja przebiega tak, jak opisywałem to w poprzedniej części kursu. Po tym miłym wstępie przechodzimy do właściwej części tego kursu, a mianowicie... do obsługi wejść mikrokontrolera.

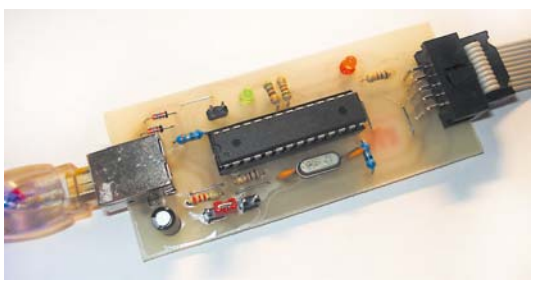
Jak sprawić, by nasz mikrokontroler zareagował na sygnały wejściowe...

W poprzedniej części udało nam się wysterować migającą diodę. Na początku tej części kursu udało nam się uruchomić nowy programator i przetestować jego działanie. Jednak migająca dioda nie doprowadzi nas zbyt daleko w dziedzinie robotyki. Wiemy, jak sprawić, by „coś" zadziałało. Ale jak uzależnić wyjście od sygnałów wejściowych, np. od przycisku start na płycie robota? Do tego, mój Drogi Czytelniku, posłużymy się pewną zdolnością mikrokontrolera… jest nią programowalny

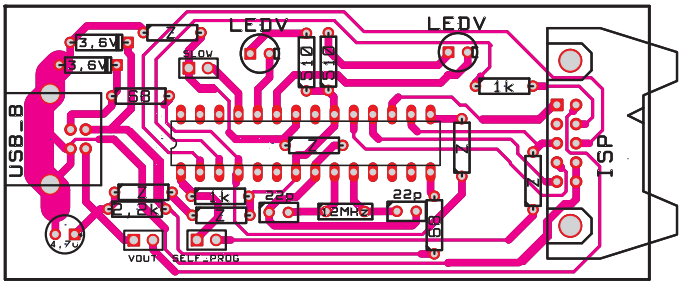


Rys. 2

Fot. 1



Rys. 1



układ IO. Każdy mikrokontroler ma kilka lub więcej nóżek, pełniących rolę portów wejścia-wyjścia. W większości wypadków w układach AVR porty (piny) podzielone są po 8 (ze względu na architekturę układu). W mikrokontrolerach AVR oznaczane są jako PORTA, PORTB, itd. Poszczególne „piny" portu oznaczane są numerami od 0 do 7. Reasumując, PORTA posiada zdefiniowane wartości o oznaczeniach PORTA.0 do PORTA.7. Każdemu z tych wyjść może też być przypisana możliwość pracy jako wejście. Wtedy w programie posługujemy się nie zwrotem PORTA.7, tylko PINA.7. Proste, nieprawdaż? Aby port mógł prawidłowo pracować jako wejście lub jako wyjście, należy odpowiednio zdefiniować odpowiedni rejestr kierunku pracy portu w wypadku PORTUA będzie to rejestr DDRA. O tym też pisałem w poprzedniej części kursu, dlatego nie będę się rozpisywał niepotrzebnie na ten temat ponownie. Następną ważną informacją jest to, że jeżeli za pomocą rejestru DDRA zdefiniujemy PORTA jako wejścia, to mamy też dodatkową możliwość ustawić (włączyć) rezystory podciągające do plusa zasilania, tzw. PULLUP-y. Rozszerza to możliwości pracy portu - wtedy w stanie spoczynku takie podciągnięte wejście ma stan wysoki (1) i zwarcie do masy zmienia stan na niski (0). Pullup aktywujemy, ustawiając PORTA w stan wysoki, np. jeżeli zdefiniujemy PINA.7, to odpowiadający mu pullup włączymy ustawiając PORTA.7 w stan wysoki.

Mam nadzieję, że jest to zrozumiałe. Pamiętam, jak na początku miałem z tym problemy. Często zastanawiałem się, dlaczego nie działa mi wyjście lub dlaczego ten procesor nie reaguje na zwarcie do masy odpowiedniego pinu. Po czym łapałem się na tym, że zapomniałem ustawić rejestr DDRD.

No cóż, początki zawsze są trudne. Tyle tytułem wstępu. Teraz pora wykorzystać naszą wcześniejszą wiedzę. Sprawmy, żeby nasz układ stał się bardziej podatny na sugestie. Proszę, stwórz nowy projekt, tak samo jak poprzednim razem, ale w treści wpisz ten program (gotowe listingi znajdziesz w Elportalu: *www.elportal.pl/robot*):

```
#define F_CPU 8000000UL
#define cbi(add,bit) ((add) &= ~(1 << bit));
#define sbi(add,bit) ((add) |= (1 << bit));
#define Led PORTB
#include „stdio.h"
#include „stdlib.h"
#include <avr/io.h>
#include <util\delay.h>
#include <avr/interrupt.h>

void migaj_led()
{
Led&=~_BV(2);
_delay_ms(200);
Led|=_BV(2);
_delay_ms(700);
}

void migaj_szybko()
{
cbi(PORTB,2);
_delay_ms(20);
sbi(PORTB,2);
_delay_ms(70);
}

int main(void)
{
sbi(DDRB, 2);
cbi(DDRB, 3);

cbi(PORTB,2);

do
{
if (bit_is_set(PINB,3)) migaj_led();
if (bit_is_clear(PINB,3)) migaj_szybko();
} while (1);
}
```

Skompiluj program, czyli naciśnij przycisk F7 na klawiaturze. Jeżeli wszystko jest ok i nie ma żadnego błędu, powinieneś otrzymać komunikat, że jest 0 błędów i 0 ostrzeżeń. Jeżeli tak nie jest, skontroluj program raz jeszcze i ponów próbę. Program jest sprawdzony i działa, więc jeżeli coś jest nie tak, to sprawdź ponownie cały proces tworzenia nowego projektu.

Gdy wszystko jest OK, wgraj plik *.hex z nowego projektu do mikrokontrolera, oczywiście korzystając z nowego programatora. No może niekoniecznie... wgraj za pomocą tego, co masz do dyspozycji. Jeżeli wszystko poszło dobrze, to obecnie na twojej płytce „mózgu" robota dioda LED miga powoli. I wygląda jakby nic się nie działo i nic się nie zmieniło od poprzedniej wersji programu. Ale naciśnij i trzymaj przycisk Start. Jeśli wszystko jest ok, to dopóki trzymasz przycisk, dioda LED miga bardzo szybko. Zobacz teraz, co się stanie, jeżeli puścisz przycisk. Dioda nie świeci, tylko znów miga powoli. Wciśnij znów przycisk. Dlaczego tak się dzieje? Przeanalizujmy nasz program, lecz tym razem skupmy się tylko na tych szczegółach, które nie zostały omówione w poprzednim odcinku, czyli obsłudze sygnału wejściowego.

Do dyspozycji mamy dwa proste polecenia:

```
if (bit_is_set(PINB,3)) migaj_led();
```

Sprawdza ono, czy zadana FLAGA (w naszym wypadku PINB,3) jest w stanie wysokim, jeżeli tak, polecenie if wykonuje to, co znajduje się bezpośrednio za nim. Jeżeli chcielibyśmy, by było to kilka poleceń trzeba je umieścić w {}

```
if (bit_is_clear(PINB,3)) migaj_szybko();
```

Natomiast to polecenie sprawdza, czy zadana FLAGA przyjęła wartość „0" (czy jest w stanie niskim). I ot cała filozofia.

Jednakże, żeby to wszystko zadziałało poprawnie, musimy odpowiednio zdefiniować rejestr kierunkowy portu B. Czyli użyjemy następującego polecenia:

```
cbi(DDRB, 3);
```

Zgodnie z tym, czego dowiedziałeś się w poprzedniej części kursu, ustawienie bitu w tym rejestrze ustawia odpowiadający pin jako wyjście, natomiast skasowanie powoduje, że odpowiedni pin staje się wejściem. Oczywiście wewnątrz warunku ***if*** możemy stosować operatory logiczne, np. można sprawdzać dwa lub więcej warunków. Ważne jest to, że polecenie if sprawdza, czy to, co jest wewnątrz nawiasu, jest prawdą czy fałszem (>0 lub 0).

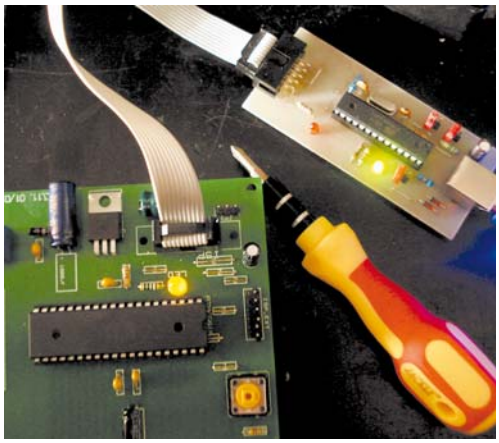
Na **fotografii 2** widać moje stanowisko badawcze. Jak widzisz, wcale nie potrzeba wiele miejsca do zabawy z robotami. Na **fotografii 3** chciałem przedstawić proces wgrywania programu naszym programatorem do płytki robota. Jednakże proces ten trwa tak szybko, że zdjęcie nie wyszło tak, jak chciałem.

Przerwania – cóż to takiego...

Skoro potrafimy już sterować wyjściami i potrafimy sprawdzić stan wejść, trzeba iść dalej. Zgodnie z wcześniejszą obietnicą, zajmiemy się kilkoma ważnymi rejestrami oraz obsługą przerwań. Co to są przerwania? Ciężko to wytłumaczyć tak na szybko, ale załóżmy, że program zajmuje się cały czas wykonywaniem procedury migania diody LED oraz czeka na naciskanie przycisku. Jest to najprostsza możliwość, jaką do tej pory poznaliśmy. A co, jeżeli chcemy, by program realizował kilka funkcji z pozoru jednocześnie? Na razie nie mamy robota, ale załóżmy, że nasz robot jedzie i odlicza jakieś stałe czasowe, by jechać po określonej trajektorii. np. zadamy mu jazdę do przodu przez kilka sekund, potem obrót w lewo i znów do przodu. Nic się nie stanie, dopóki nasz waleczny robot... nie spadnie ze stołu. I tu przydają się przerwania. Ponieważ program może „w tle" odliczać czas, ale niejako „w gotowości czekać" na sygnały ważnych czujników linii (podłoża), w razie wykrycia braku podłoża lub najechania na „białą" linię, program, który w zasadzie robi coś innego, otrzyma

Fot. 2

Fot. 3



„pilny sygnał zewnętrzny” i wykona odpowiednią czynność, zapobiegając uszkodzeniu robota lub przegraniu walki, jeżeli nasz robot zjechałby z ringu. Najprostszym przykładem niech będzie poniższy przykład wykorzystujący przerwanie od tzw. Timera.

A teraz kolejny program. Proponuję każdy nowy program pisać jako nowy projekt lub plik, by nie stracić czegoś ważnego z poprzednich prac.

```
#define cbi(add,bit) ((add) &= ~(1 <<
bit));
#define sbi(add,bit) ((add) |= (1 <<
bit));

#define F_CPU 8000000UL
#include „stdio.h”
#include „stdlib.h”
#include <avr/io.h>
#include <util\delay.h>
#include <avr/interrupt.h>
#define  Led  PORTB
                        //Do tego pinu
podlaczona jest dioda LED2

void migaj_led(int licz)
{
int i=0;
while(i<=licz){
Led&=~_BV(3);
_delay_ms(200);
Led|=_BV(3);
_delay_ms(700);
++i;
}
}

SIGNAL (SIG_OVERFLOW0)
{
__asm(„cli”);
cbi(PORTB,2);
_delay_ms(60);
sbi(PORTB,2);
_delay_ms(30);
}

int main(void)
{
sbi(DDRB, 2);
cbi(DDRB, 3);
sbi(PORTB,3);

migaj_led(4);

TCCR0=0x03;       // prescaler 64
TCNT0=0x00;       // wartośc początkowa
zliczania = 0
TIMSK=0x01;       // Timer/Counter0
Overflow interrupt is enable
//__asm(„sei”); //zezwolenie na prze-
rwania globalne

do
{
if (bit_is_set(PINB,3)) {__
asm(„sei”);};
if (bit_is_clear(PINB,3)) {__
asm(„cli”); cbi(PORTB,2);};
}
while(1);
}
```

Skompiluj, proszę, standardowo ten program i wgraj do mikrokontrolera. A teraz zastanówmy się i przeanalizujmy, co się tu dzieje i dlaczego.

Przede wszystkim w podanym przykładzie po raz pierwszy przydała się nam biblioteka *#include <avr/interrupt.h>*, we wcześniejszych przykładach dodawana tylko dlatego, by stworzyć pewien szablon. Następnie pomijając całą masę poleceń wcześniej omówionych docieramy do *SIGNAL (SIG_OVERFLOW0) { }* i pierwsze pytanie: a cóż to takiego? Bardzo upraszczając jest to z góry zdefiniowany podprogram, mający na celu zadziałać w wypadku przekroczenia wartość licznika (TIMERA) 0. I tyle. Po prostu załóżmy, że licznik liczy od 0 do FF (w wartościach haksadecymalnych), to znaczy, że nasz SIG_OVERFLOW0 zostanie uruchomiony, jeżeli licznik doliczyłby do FF i zwiększył swój stan. Wtedy też zostałoby wygenerowane przerwanie OVERFLOW. Wewnątrz procedury pojawia się nowa komenda: *__asm(„cli”);*

Co ona robi? Otóż wyłącza obsługę przerwań. Po co? Pamiętajmy, że przerwania mogą przerywać pracę programu i wykonywać niezależne fragmenty programu. I tu może powstać kłopot, jeżeli wykonanie danej czynności zajmuje więcej czasu niż wykonanie innego przerwania. Wtedy możemy mieć bardzo niestabilny program. Trzeba o tym pamiętać i jeżeli zależy nam na czymś, to właśnie poleceniem *__asm(„cli”);* wyłączajmy obsługę przerwań na czas potrzebny do wykonania zadanego fragmentu programu, a poleceniem *__asm(„sei”);* włączajmy obsługę, gdy już procedura się skończy.

Następnie natrafimy na kilka nowych pojęć:

```
TCCR0=0x03;       // prescaler 64
TCNT0=0x00; // wartośc początkowa
                          zliczania = 0
TIMSK=0x01;       // Timer/Counter0
          Overflow interrupt is enable
```

Każde z nich definiuje parametry określonego rejestru, który odpowiada za konfigurację i poprawną pracę specyficznych obszarów mikrokontrolera. Tutaj mamy przykład konfiguracji licznika TIMER0. Dlatego używamy rejestru TCCR0, TCNT0. Rejestr TIMSK zawiera w sobie „przełącznik”, który włącza możliwość pojawiania się w programie przerwań pochodzących od różnych sygnałów. Dokładniej tym rejestrem zajmiemy się podczas pisania programu całego robota. A na razie sugeruję, byś pobrał z Internetu plik *.pdf z opisem Twojego mikrokontrolera. Tam opisane są bardzo dokładnie wszystkie rejestry. W przykładzie pojawia się słowo *prescaler* – jest to wewnętrzny układ, który służy do „zmniejszenia” prędkości z jaką timer liczy. W tym przykładzie wartość prescalera wynosi 64, co mniej więcej znaczy, że licznik zlicza z prędkością 8 000 000/64 razy na sekundę, czyli osiąga 125 000 zliczeń w ciągu sekundy – jeżeli założymy, że liczy od 0 do 254, to uzyskamy około 490 impulsów OVERFLOW0.

Mam nadzieję, że reszta programu jest już zrozumiała po takim wyjaśnieniu.

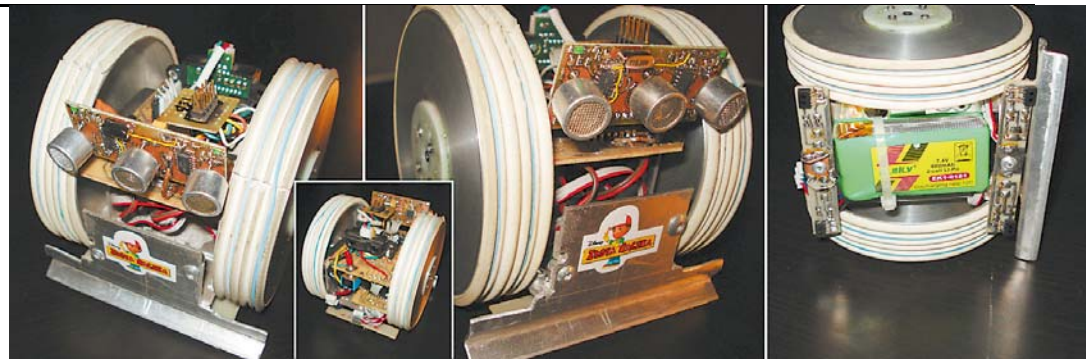
Po przeanalizowaniu powyższych programów potrafimy już wysterować wyjście, sprawdzić jego stan, a także potrafimy realizować kilka czynności niejako jednocześnie, korzystając z systemu przerwań mikrokontrolera.

Bardzo zachęcam do eksperymentowania samodzielnie z przerwaniami i programem. Nie bójcie się mikrokontrolera – programowo nie da się go uszkodzić, no chyba, że zaczniecie bawić się fusebitami, a tego na razie nie polecam. Tyle na dziś, a następny odcinek zaczniemy od PWM. Potem omówimy kolejny moduł, a będzie nim moduł wzmacniaczy do czujników odbiciowych, czyli czujników „koloru”. Służą one do tego, by nasz dzielny zawodnik nie wyjechał z dohyo podczas walki sumo lub po prostu by nie spadł np. ze schodów. Oczywiście będę dalej męczył Was programowaniem, a dokładniej jego podstawami. To w sumie jest dużo ważniejsze od samej elektroniki robota. Jeżeli bowiem uda mi się przekazać Wam to, co chcę, to sprawicie, że każdy z robotów będzie miał swój unikalny „charakter”, pomimo podobnej konstrukcji elektroniki czy też mechaniki. Bowiem to właśnie program, który stworzysz, stanowi o możliwościach robota. W każdym razie ja postaram się przekazać podstawy, a reszta to już tylko Twoje chęci i ciekawość. Zapraszam do lektury następnego odcinka.

I jeszcze raz proszę o maile z wszelkimi uwagami na temat tego cyklu.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

kit 2935 AVT



Robot mobilny – krok po kroku

część 4

Obiecałem, że w tej części kursu zajmiemy się specjalizowaną częścią struktury mikrokontrolera, jaką jest PWM. To bardzo pomocna rzecz szczególnie w przypadku, gdy zależy nam na sterowaniu mocą silników czy jasnością diod LED. Sam PWM nie jest „częścią" mikrokontrolera, ponieważ jest to skrót z języka angielskiego oznaczający Pulse Width Modulation (modulacja szerokości impulsu). Zasada działania PWM jest bardzo prosta. Polega ona na generowaniu fali prostokątnej o określonej częstotliwości, ale o zmiennym wypełnieniu. Dzięki późniejszej obróbce uzyskujemy uśrednioną wartość w zakresie od 0 do 5V. Zasadę działania obrazują **rysunki 1–3**. Więcej informacji, jeżeli kogoś interesuje ten temat, można znaleźć w nocie aplikacyjnej ATmegi oraz np. na stronie: *http://en.wikipedia.org/wiki/Pulse-width_modulation*.

Ważne jest to, że generator PWM to fragment mikrokontrolera, działający w zasadzie samodzielnie, poza głównym programem. Raz ustawiony przez program, pracuje do momentu wyłączenia. Możemy zmieniać wartość wypełnienia przebiegu, wpisując do odpowiedniego rejestru wartość liczbową odpowiadającą temu, co chcemy osiągnąć.

Niezbędne rejestry różnią się w zależności od typu mikrokontrolera, jakiego chcemy użyć. Ale ponieważ w naszym projekcie używana jest ATmega16 lub ATmega32, te rejestry są takie same. Nie powinno zatem być kłopotów z ich konfiguracją. Rejestry, których będziemy używać, są następujące: TCCR1A, TCCR1B, OCR1AL, OCR1AH, OCR1BL, OCR1BH.

Aby nasz PWM zadziałał, w programie przed main{} należy dodać poniższy podprogram:

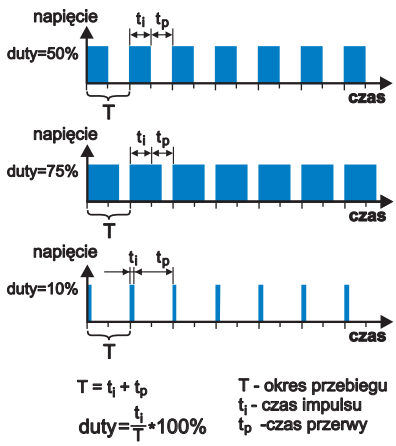
```
void init_PWM(void) {
sbi(DDRD,4);
sbi(DDRD,5);
TCCR1A|=_BV(COM1A1);
TCCR1A|=_BV(COM1B1);
TCCR1A|=_BV(WGM10);
TCCR1B|=_BV(WGM12);
TCCR1B|=_BV(CS11);
OCR1AH=0x00;
OCR1AL=0;
OCR1BH=0x00;
OCR1BL=0;
}
```

Ustawiam bity zgodnie ze specyfikacja ATmegi podaną w datasheet.

Tryb pracy: FAST PWM, aktywne wyjścia OC1A and OC1B, Counter1 Prescaler 8, Mode 5 – Fast PWM, 8bit, TOP=0x00FF, Set OC1A/OC1B on Compare Match, Clear at TOP – nie będę się rozpisywał, co oznaczają poszczególne rzeczy, gdyż wszystkie wyjaśnienia znajdują się w oryginalnym datasheet ATmela, dostępnym na ich stronie internetowej.

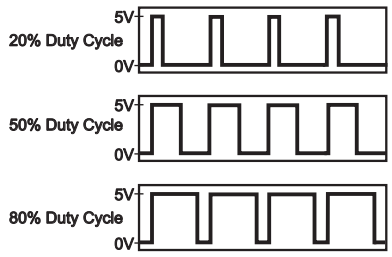
Rejestry te odpowiadają za konfigurację pracy wyjść PWM procesora. Dzięki poprawnym ustawieniom PWM będziemy mieli pełną kontrolę nad prędkością jazdy robota. Za pomocą dwóch zmiennych OC1AL i OC1BL będziemy regulować prędkość jazdy robota. Rejestry OC1AH i OC1BH w chwili obecnej nas nie interesują, gdyż PWM pracuje w trybie 8-bitowym. PWM-y wykorzystamy później w końcowym programie sterującym robotem. Na chwilę obecną, drogi Czytelniku, w pętli main wpisz *init_PWM();* Po skompilowaniu programu powinno być wszystko OK. Tu ważna uwaga: procedurę *init_PWM* dopisujemy do poprzedniej działającej wersji programu. Bo to jest tylko dodatek, nie cały program. Oczywiście można napisać to ładniej, ale chodziło mi o to, by było widać, co robię. Proponuję przez dłuższy czas pobawić się tymi poznanymi właśnie możliwościami mikrokontrolera. Do wyjść PWM można podpiąć np. diodę LED i bawiąc się rejestrami OC1A i B, regulować jasność świecenia. Pamiętajmy, że procesor nie gryzie, nie zróbmy tylko zwarć na wyjściach. LED sugeruję podpinać z rezystorem 220Ω w szeregu – ogranicza to prąd płynący w obwodzie i zabezpiecza „mózg" naszego robota przed uszkodzeniem.

Po tym skrótowym opisie PWM sądzę, że moduł „mózgu" ma coraz mniej tajemnic przed nami. Wiemy już, jak sprawdzić stan wejść, wiemy, jak wymusić określony stan wyjść, znamy już podstawy posługiwania się przerwaniami, a teraz wiemy, jak sprawić, by na wyprowadzeniach mikrokontrolera pojawił się sygnał PWM.

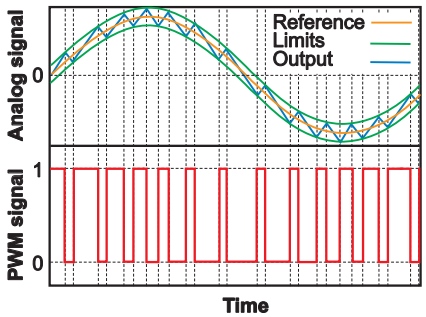


Rys. 2

Rys. 3



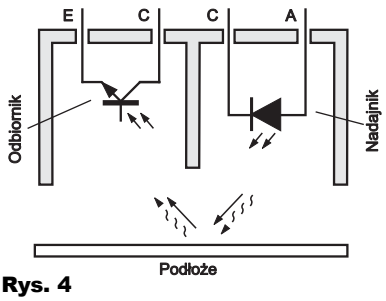
Rys. 1



Oczy robota, czyli jak sprawić, by robot reagował na otaczający go świat

Każda konstrukcja w jakikolwiek sposób zautomatyzowana zawiera czujniki „środowiskowe". Czy będą to czujniki optyczne,

Fot. 1



Rys. 4

czy mechaniczne, czy jakiekolwiek wykorzystujące inne zjawiska fizyczne – służą do przekazania informacji ze świata „żywego" do mikrokontrolera. Czujniki mogą podawać wartości analogowe (o zmiennej wartości) lub bardziej „cyfrowe", to znaczy *jest* lub *nie ma*. Najprostszym przykładem czujnika „analogowego" jest potencjometr. Najprostszym przykładem czujnika cyfrowego jest przycisk (zwierny lub rozwierny). Nasz robot będzie wykorzystywał dwa podstawowe rodzaje czujników. Będą to: optyczne czujniki „koloru" do detekcji podłoża oraz dwa czujniki mechaniczne tzw. wąsy (mikroprzełączniki z dźwignią). Nic szczególnego, ale dzięki temu nie spadnie ze stołu ani nie zaprze się o ścianę. Jeżeli będzie zainteresowanie innymi czujnikami, to zajmiemy się nimi. Ja w swoich konstrukcjach używam czujnika ultradźwiękowego opracowanego przez studentów zrzeszonych w KONAR. Jest super do zabawy i tani w wykonaniu. No i jego dokumentacja jest darmowa. Ale wracając do tematu... Czujników w postaci przycisku z dźwigienką, takich jak na **fotografii 1**, nie muszę przedstawiać i opisywać. Wystarczy dobrać gabarytem do konstrukcji robota. Natomiast czujniki odbiciowe (optyczne lub koloru, jak zwał, tak zwał) wymagają „dopieszczenia tematu". I nimi zajmiemy się troszkę dokładniej.

Sam czujnik odbiciowy w zasadzie to połączenie elementu „świecącego" (najczęściej w podczerwieni) oraz elementu, który jest wrażliwy na tę barwę. Elementy te są oddzielone od siebie przegrodą nieprzepuszczającą „światła" z elementu emitującego wprost do elementu detekcyjnego. Jedyna możliwość to odbicie promieniowania od przeszkody (w naszym wypadku podłoża). Elementem, który odbiera odbite promieniowanie, najczęściej jest fototranzystor lub fotodioda. Jako że w EdW było na temat transoptorów mówione wiele razy, nie będę tłumaczył tego od nowa. Dla wyjaśnienia organoleptycznego zamieszczam szkic transoptora odbiciowego – **rysunek 4**, który wyjaśni, mam nadzieję w 100%, co i jak.

Schemat modułu, jaki zostanie przez nas wykorzystany w robocie, to bardzo prosta aplikacja wzmacniacza operacyjnego pracującego w zasadzie jako komparator. Ale jeżeli ktoś będzie chciał, zawsze może zbudować od nowa i wykorzystać inny układ detekcyjny lub przebudować ten proponowany przeze mnie. Układ detektora wykorzystuje bowiem piny portu A ATmegi, a co za tym idzie, można zdefiniować go jako przetworniki analogowo-cyfrowe, a nie tylko jako detekcję logicznych 0 i 1.

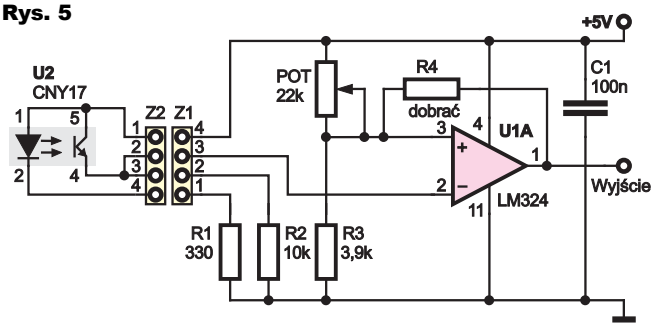
Schemat jednego czujnika znajduje się na **rysunku 5**. Do regulacji progu zadziałania służy potencjometr POT. Jest to oczywiście jeden z czterech identycznych detektorów o identycznych schematach. Potencjometr POT reguluje wszystkie wzmacniacze jednocześnie. Innymi słowy, wszystkie wejścia nieodwracające połączone są razem.

Zasada działania jest bardzo prosta i myślę, że nie wymaga specjalnego tłumaczenia. W ramach projektu udostępniam dwa projekty płytek wykonanych w KiCAD-zie. pierwszy – są to same otwory pod piny łączące płytki, dzięki którym możesz, drogi Czytelniku, zaprojektować własny moduł koloru.

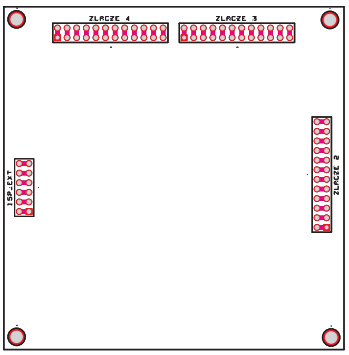
Drugi projekt – to moduł detektora koloru mojego pomysłu z wyprowadzonymi złączami do podlutowania dowolnego czujnika, np. TCRT 1000 lub CNY70. Jedynie, co trzeba, to dobrać wartość rezystora w sprzężeniu zwrotnym wzmacniacza operacyjnego.

Wzory płytek drukowanych znajdują się na **rysunkach 6** i **7**. Jak widać, płytki są banalnie proste. Złącza rozmieszczone prawie w rogach płytki to złącza do przylutowania transoptorów. Złącze 3-pinowe znajdujące się w górnej części płytki to złącze do podłączenia „wąsów". Potencjometr do regulacji specjalnie znajduje się na brzegu płytki. Dzięki takiemu umieszczeniu będzie łatwy dostęp podczas prac testowych. Jak wspomniałem wcześniej, zachęcam do zbudowania własnego modułu o większych możliwościach. Jednakże do tego co chcemy osiągnąć, zaproponowany moduł będzie wystarczający. Na **fotografii 2** prezentuję moduł wykonany przeze mnie za pomocą termotransferu. Montaż płytki wykonujemy oczywiście po swojemu lub według mojej metody opisanej w części pierwszej cyklu. Uwaga, co do modułu. Rezystory oznaczo-

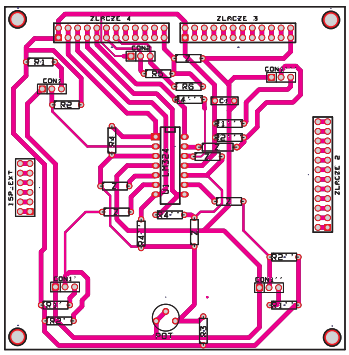
Rys. 5

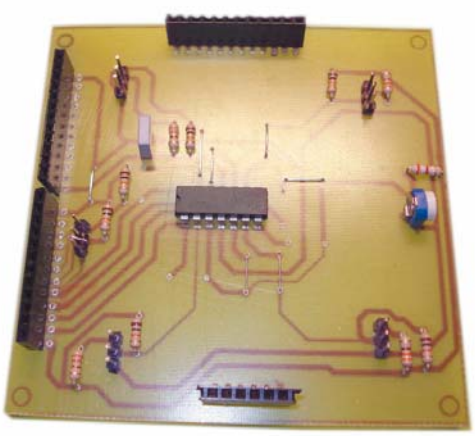


Rys. 6 skala 50%



Rys. 7 Skala 50%





Fot. 2

ne DOBRAĆ trzeba dobrać doświadczalnie. Spotkałem się w swoich konstrukcjach, że poprawne efekty uzyskiwałem, stosując wartości 510kΩ, ale najlepiej zacząć od wartości dużych, np. 4,7MΩ, lub sprawdzić działanie, nie montując w ogóle tych rezystorów.

W następnym odcinku dalej będziemy się zajmować programowaniem i ożywianiem naszego robota. Sugeruję zaopatrzyć się w płytkę modułu i czujniki, gdyż postaramy się je ożywić. Czujniki do wyboru. Ja stosuję TCRT1000, ale wiem, że z powodzeniem można wykorzystać CNY70 i wiele innych dostępnych na rynku.

Odpowiedzi na pytania Czytelników

W poprzednich częściach nie umieściłem pewnych informacji, gdyż uważałem, że nie są one aż tak istotne albo po prostu rutynowo uznałem je za oczywiste. Na prośbę Czytelników wyjaśniam:

1) Na schemacie modułu „mózgu" umieszczony jest element L1 w obwodzie zasilania części analogowej mikrokontrolera – jest to dławik o indukcyjności 10µH (taki jaki jest zalecany w nocie aplikacyjnej ATmegi).
2) Bardzo ważną rzeczą jest, by domyślny katalog instalacji programu WinAvr był następujący: **C:\\winavr**. Jeżeli będzie inny, pojawią się kłopoty z integracją AVRStudio z WinAVR.
3) Projekt programatora USB nie jest mojego autorstwa, jest to darmowy projekt, opracowany i sprawdzony przez wielu użytkowników. Można go kopiować i na 100% działa. Zupełnie inną sprawą jest jego oprogramowanie, gdyż oficjalne wsady i sterowniki nie będą działały z AVRStudio. Należy wgrać wsad i zainstalować sterowniki AVRDoper (w trzeciej części kursu podałem odpowiednie linki), tu mała uwaga – nie działa jedna z diod LED, a druga oznacza programowanie, reszta w zasadzie bez zmian. Oczywiście wsad wgrywamy ten odpowiadający AVRUSB.

Proszę pisać jak zwykle na mój e-mail. Jeżeli tylko będę w stanie – będę wyjaśniał, podpowiadał i pomagał.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

Wykaz elementów

Moduł czujników

R1,R1',R1'',R1''' 330Ω
R2,R2',R2'',R2''' 10kΩ
R3 ..... 3,9kΩ
R4,R4',R4'',R4''' ..... 500kΩ...6MΩ * dobrać (lub brak)
POT ..... 22kΩ PR
C1 ..... 100nF
U1 ..... LM324
U2,U2',U2'',U2''' ..... transoptor odbiciowy (CNY17, TCRT1000, itp.)
Z1,Z2 ..... złącza szpilkowe lub inne (na płytce są 3-pinowe)
Z ..... zworka
CON1,CON1',CON1'',CON1''',CON2,złącze2,złącze3,złącze4,ISP_EXT ..... złącza szpilkowe, raster 2,54mm (1-rzędowe)

Komplet podzespołów z płytką dostępny jest w sieci handlowej AVT jako kit szkolny AVT-2935/1.

kit 2935 AVT

Robot mobilny – krok po kroku część 5

W tej części zajmiemy się uruchomieniem i sprawdzeniem modułu „wzroku” naszego robota. Następnie podłączymy moduł do pozostałych części „kanapki”. W ten sposób mamy kompletny elektroniczny system sterowania. Robot składa się z modułów, by można było zastępować je innymi. Na przykład, zamiast stosować driver silników na użytym małym układzie, można wykorzystać układ o większej mocy. Dzięki temu każdy może swojego robota rozbudować.

„Oczy” – moduł czujników środowiskowych

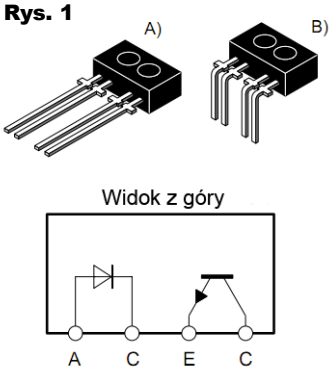
Jak pisałem w poprzedniej części, trzeba się zaopatrzyć w czujniki odbiciowe. Ja posiadam TCRT1000, dostępne w ofercie handlowej TME. Nie są one może najtańsze, ale kiedyś kupiłem kilkanaście i ich używam. Można też zastosować czujniki CNY70.

Pierwszy to TCRT1000 – opis jego wyprowadzeń oraz wygląd pokazany jest na **rysunku 1**. Drugi to CNY70 – **rysunek 2**. W zasadzie można je przylutować bezpośrednio do płytki, ale można zrealizować różne konstrukcje na różnych podzespołach, toteż nie ma z góry narzuconego jedynie słusznego rozmieszczenia tych czujników. Powinny one być umieszczone jak najbliżej krawędzi konstrukcji mechanicznej robota. Jednocześnie muszą być na określonej wysokości nad podłożem (tę odległość trzeba dobrać doświadczalnie – tu właśnie pomocne okaże się uruchamianie tego modułu z dodatkowymi diodami LED). Ja umieściłem czujniki na końcach odcinków przewodu. A diody LED umieszczę w górnej części platformy na stałe. Będę mógł obserwować, jak robot „reaguje” na podłoże.

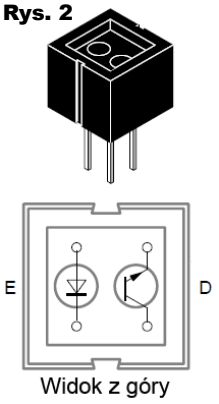
Jeszcze słowo o czujnikach odbiciowych: każdy, kto ma diodę świecącą (IRED, ale można też wypróbować LED) i fototranzystor, może samodzielnie wykonać takie czujniki. Po prostu trzeba jeden z elementów umieścić w odcinku rurki termokurczliwej (nieprzepuszczającej światła i podczerwieni). Dzięki takiemu rozwiązaniu można uzyskać bardzo interesujące czujniki. Widziałem niejedną konstrukcję, w której zostały z powodzeniem wykorzystane właśnie takie własnej roboty, czujniki.

Należy pamiętać, że czujniki odbiciowe, z uwagi na sposób pracy, muszą być umieszczone na pewnej wysokości nad podłożem. Inaczej są „ślepe”. W naszym module czujniki współpracują ze wzmacniaczem – komparatorem, a właściwie przerzutnikiem Schmitta – patrz rysunek 5 w poprzednim odcinku w EdW 6/2020. Nie interesuje nas analogowa wartość (czyli upraszczając... odcień podłoża). Tak ustawiamy potencjometr czujnika, by brak podłoża, lub czarne podłoże, dawało jeden stan wyjścia i by stan ten zmieniał się po wykryciu białego podłoża. To będzie pomocne dla osób pragnących zbudować robota do „walki” na ringu. Do pierwszych testów modułu czujników, bez uruchamiania całego robota, przyda się oddzielny zasilacz 5V, który zasili tylko moduł czujników. Wtedy do wyjść wzmacniaczy operacyjnych trzeba dołączyć diody sygnalizacyjne LED + rezystory szeregowe np. 220Ω. Na **fotografii 1** przedstawiony jest mój osobisty warsztat testowy, na którym widać, jak rozwiązałem (testowo) podłączenie diod LED oraz podłączenie transoptora odbiciowego. Szczegółowo podłączenie transoptora jest widoczne na **fotografii 2**. Płytka wyposażona jest w cztery transoptory i 2 wejścia do przycisków no/nc, których można użyć jako „wąsy” – to, czy ich użyjesz, czy nie, pozostawiam do wyboru Tobie, Drogi Czytelniku.

Rys. 1



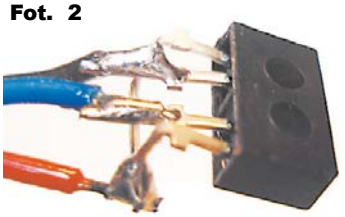
Rys. 2



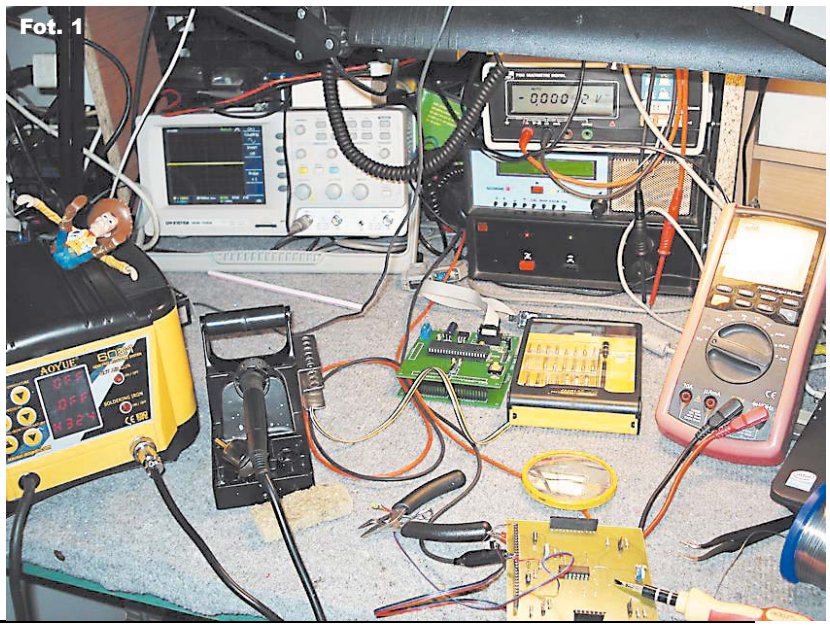
Ożywiamy oczy robota... czyli programowania ciąg dalszy

By to wszystko zadziałało, należy wykorzystać to, czego nauczyliśmy się w poprzednich częściach kursu, czyli umiejętność obsługi kilku rejestrów i obsługi wejść i wyjść. Płytki tworzą „kanapkę” w taki sposób, by odpowiednie piny doprowadzały do procesora odpowiednie sygnały. I tak „oczy” robota podłączone są do wejść PORTUA mikrokontrolera ATmega. Dlaczego... ano dlatego, że są to porty, które mogą też pracować jako przetworniki A/C (analogowo-cyfrowe). Tutaj nie będziemy ich wykorzystywać w roli przetworników A/C, a tylko jako wejścia dwustanowe. Ze wzmacniaczy operacyjnych LM324 otrzymujemy „1” logiczną, jeżeli czujnik odbiciowy jest nad czarnym tłem, lub „0” logiczne, jeżeli jest nad tłem białym. Dzięki czemu mamy bardzo prostą detekcję wjechania na białą linię.

Fot. 2



Fot. 1



Listing 1

```
while (1)
{
if (bit_is_clear(PINA,0)||bit_is_clear(PINA,1)||(bit_is_clear(PINA,2)¬
||bit_is_clear(PINA,3)) {cbi(PORTB,2);}
else
sbi(PORTB,2);
};
```

Jeżeli ktoś chce zbudować robota typu LF (line follower), wystarczy nieznacznie zmienić program, by uzyskać robota „trzymającego się" białej lub czarnej linii.

Czy zaproponowane rozwiązanie jest wystarczająco dobre, czy też ma mankamenty?… Hm, jest na pewno proste i dość skuteczne. Zawsze twierdziłem, że należy uczyć się na własnych błędach, podpierając się pomocną dłonią. I tak też jest w tym przypadku. Mój projekt daje wiele dodatkowych możliwości każdemu, kto chce mieć więcej frajdy. W przyszłości możesz sam zmienić obwody elektroniczne i program czujnika. Cały ten cykl powstał po to, by zachęcić do samodzielnych działań. Ja chętnie pomogę i podpowiem.

Wracamy do głównego wątku: aby czujniki działały, musimy zdefiniować cały PORTA (lub przynajmniej odpowiednie piny) do pracy jako wejścia, czyli należy wpisać w programie polecenie *DDRA=0x00;* lub w postaci bitowej *DDRA=0b00000000;* z poprzednich części wiadomo dlaczego. By czujniki zadziałały, nie musimy w zasadzie niczego więcej definiować. Ale należałoby je wykorzystać w programie, by powodowały reakcję robota. Proponuję w celach testowych napisać taką pętlę (**listing 1**) i umieścić w *main* (najlepiej na końcu).

Sugeruję, byś napisał program samodzielnie, ponieważ pętla ta jest banalnie prosta. Jej działanie sprowadza się do zaświecenia diody LED na płytce „mózgu" w czasie, gdy jest aktywny któryś z czujników odbiciowych. A gaśnie w zasadzie „natychmiast" po ustąpieniu „zjawiska", czyli najprościej obrazowo tłumacząc – jeżeli którykolwiek z czujników jest na białej linii, to świeci dioda LED na płytce „mózgu".

Oczywiście, aby to zadziałało, należy prawidłowo zdefiniować porty, jak pisałem w poprzednich częściach cyklu. Podaję listę wejść użytych w mojej wersji modułu „oczu".

PINA0, PINA1, PINA2, PINA3 – to wejścia z komparatora LM324. Kolejność dowolna, między innymi ze względu na otwartość konstrukcji. Wejścia PINA4 i PINA5, które posłużą docelowo jako wejście „wąsów", są „podciągane" do +5V za pomocą dodatkowych rezystorów 10kΩ.

Po napisaniu prawidłowo programu powinieneś uzyskać coś mniej więcej w stylu z **listingu 2**.

Chcę Ci teraz pogratulować posiadania działającej części elektronicznej robota. Pozostaje jeszcze napisanie oprogramowania i zbudowanie części mechanicznej robota. W chwili obecnej, Drogi Czytelniku, jesteś w stanie samodzielnie napisać program sterujący do Twojego robota, by już samodzielnie się on poruszył i odpowiednio do sytuacji zareagował.

```
#define F_CPU 8000000UL //predkosc taktowania
#define cbi(add,bit) ((add) &= ~(1 << bit));
#define sbi(add,bit) ((add) |= (1 << bit));
#include „stdio.h"
#include „stdlib.h"
#include <avr/io.h>   //adresy rejestrow
#include <util\delay.h>  //biblioteka z opoznieniami
#include <avr/interrupt.h>

int main(void)
{
DDRB=0xFF;
DDRA=0x00;
PORTA= 0xFF; // włączamy wewnętrzne pull'up
while (1)
{
if (bit_is_clear(PINA,0)||bit_is_clear(PINA,1)¬
||bit_is_clear(PINA,2)||bit_is_clear(PINA,3)) {cbi(PORTB,2);}
else
sbi(PORTB,2);
};
}
```

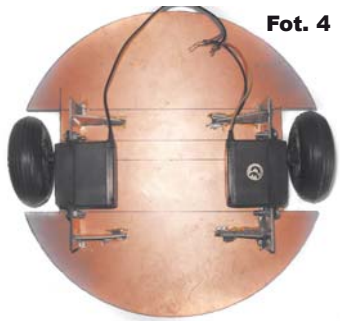
Listing 2

Długo zastanawiałem się czy umieścić w tym odcinku jakiś program robota już działającego... Doszedłem jednak do wniosku, że jeżeli ktoś budował do tej pory tego robota zgodnie z moimi wskazówkami, to nie potrzebuje programu, lecz raczej wskazówek, co do części mechanicznej. Przecież na razie nie mamy jeszcze części mechanicznej: podwozia z silnikami i kołami, więc uruchomienie programu kompletnie nic nie da. Dlatego teraz przejdziemy do świata „mechaniki".

Co zrobić, jak wykonać i z czego... budujemy konstrukcję nośną

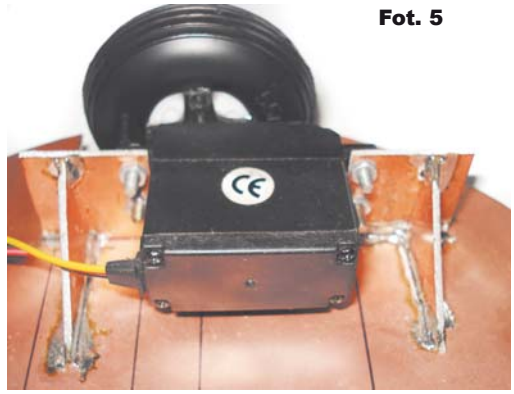
Zgodnie z tym, co pisałem na początku serii, realizacja robota wymaga od Czytelnika troszkę pomysłowości i inwencji. Nie będzie tu bowiem dokładnych planów technicznych do budowy podwozia ani kursu posługiwania się narzędziami, np. piłą, pilnikiem czy wiertarką. Chciałbym raczej zaprezentować kilka konstrukcji, które być może podsuną Ci pomysł na Twoją własną wersję podwozia robota. Po pierwsze, zastanów się, co tak naprawdę chcesz zbudować. Czy ma to być robot free-style, minisumo, sumo, line follower. Bo od tego zależeć będzie, co dalej. Następna sprawa – z czego chcesz wykonać podwozie i ile pieniędzy możesz na to przeznaczyć? Widziałem konstrukcje z napędami opartymi na poprzerabianych serwomechanizmach i na silnikach ze starych zabawek lub od magnetofonów. Uwierzcie mi, że wiele można zrobić w zasadzie ze „złomu i śmieci", pozyskiwanych za darmo.

Tak naprawdę budowa części napędowej jest ograniczona jedynie Twoją wyobraźnią. Pamiętaj tylko, że klasa *mini sumo* to 10x10cm i waga do 0,5kg (dane z KONAR-u), a *sumo* to 20x20cm i waga do 3kg (dane z CYBAIRBOT2010). Reszta, czyli zarówno *linefollower* jak i *freestyle* to konstrukcje dowolne, ale o parametrach ograniczanych przez organizatorów imprez. Wracając do moich propozycji. Na **fotografiach 3–5** pokazuję swoją wersję robota na zawody *linefollower*. Robot, jak widać, bazuje na module elektronicznym, który budujemy w tym cyklu. Nie jest jeszcze skończony, jest w trakcie budowy. Czujniki umieszczone są wszystkie z przodu na łuku. Dzięki temu robot może określić nagłe zmiany linii, nie „spadając z niej". Robota napędzają dwa serwomechanizmy przerobione tak, by kręciły się cały czas (nie ma ogranicznika



Fot. 4

Fot. 3



Fot. 5


Fot. 6


Fot. 7


Fot. 8

kąta obrotu i wyrzucona jest elektronika), opis jak wykonać taką przeróbkę znajdziesz pod znanym Czytelnikom EdW adresem: **http://www.dioda.com.pl/forum/topics20/jak-przerobic-serwomechanizmy-vt178.htm**. Oczywiście takich opisów znajdzie się więcej w Internecie, ale ten jest jednym z najlepiej napisanych.

Jak widać, całe podwozie wykonane jest z jednostronnego laminatu oraz kilku kawałków dwustronnego laminatu. Każdy elektronik ma takowy na stanie. Silniki zostały kupione kiedyś przez Internet. Koła: są to standardowe koła „piankowe", stosowane w modelach samolotów. Koszt kilka złotych za sztukę. Ale jak pisałem wcześniej, masz całkowitą dowolność. Baza tej konstrukcji to koło o średnicy 20,5cm (taki kawałek laminatu akurat miałem). Pomiędzy kołami (i silnikami oczywiście) będzie się znajdował akumulator 7,2V lub 9V (zależy, jaki będzie tańszy w zakupie). Mój robot do klasy *freestyle* będzie częściowo także bazował na module, który powstaje w tym cyklu, jednakże moduł będzie trochę zmodyfikowany. Nie wnikając w szczegóły, moduł będzie sterował napędem robota i detekcją przeszkód na trasie. Na **fotografiach 6-9** prezentuję konstrukcję opartą na… starej obudowie od centralki alarmowej i pięknych kołach zakupionych w firmie WOBIT. Serwomechanizmy napędzające tego robota to także zakup dokonany w firmie WOBIT, ale te konkretne silniki otrzymałem z Redakcji EdW. Jeżeli ktokolwiek wyrazi chęć na opracowanie podwozia jako gotowego kitu, bardzo chętnie takowy opracuję. Na razie wydaje mi się, że lepszym rozwiązaniem będzie zaopatrzenie się w mniesze koła prezentowane na **fotografii 10** (także z WOBIT-u) oraz przeznaczone do nich silniki z przekładniami. Koła te umożliwią zbudowanie robota zgodnego z klasą *minisumo*. A cała konstrukcja mechaniczna może być wtedy wykonana z płytek drukowanych!

A teraz o elastyczności: pierwotny plan był taki, aby na silnikach, które otrzymałem z EdW, zaprojektować podwozie mogące spełnić warunki uczestnictwa w zawodach *sumo*, jednakże gdy je otrzymałem, okazało się, że są o kilka milimetrów za długie by zmieścić się razem z kołami i odpowiednimi przelotkami w wymaganych w tej klasie 20cm :-(. Przyznaję, jest to mój błąd, gdyż nie uwzględniłem specjalnych przelotek, które są niezbędne do sprzęgnięcia kół z ośkami silników. Niestety nie posiadam sprzętu takiego jak obrabiarka, by odpowiednio skrócić te dystanse, toteż wykorzystałem te silniki do zbudowania robota, który będzie w zawodach uczestniczył w klasyfikacji *freestyle*. W tej klasie jedynym wymogiem jest inwencja twórcza.

Fot. 9

I co dalej?

Pamiętajcie… elektronika nie gryzie. Tu trzeba eksperymentować, oczywiście z pewną dozą ostrożności, by nie narażać się na dodatkowe i niepotrzebne koszty. Bardzo przydatne informacje uzyskacie na kilku stronach internetowych. Między innymi: **http://www.eres.alpha.pl/elektronika/readarticle.php?article_id=408**
http://www.dioda.com.pl/forum/topics33/porady-dla-poczatkujacego-robotyka-lutowanie-vt1029.htm
http://adambyw.fm.interia.pl/
http://www.konar.ict.pwr.wroc.pl/index1.php

Jeżeli będzie trzeba, zawsze pomogę i podpowiem. Piszcie jak zwykle. Następną część cyklu poświęcimy prawie całkowicie programowaniu i odpowiedziom na pytania Czytelników.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

Fot. 10

kit 2935 AVT

Robot mobilny – krok po kroku część 6

Mam nadzieję, że wcześniejsze informacje były dla Ciebie jasne. Zgodnie z obietnicą w tej części zajmiemy się głównie kwestiami programowania, a dokładniej omawiania poszczególnych fragmentów kodu.

Do testów sugeruję zafundować sobie kartkę formatu A4 z wydrukowanym czarnym prostokątem. Oczywiście trzeba zostawić około 1cm białego obwodu, by robot mógł wykryć „krawędź" ringu (doyho). Dla bardziej zamiłowanych użytkowników... możecie wykonać prawdziwe dojo dla klasy minisumo – to jest czarne koło z białym obrzeżem:
średnica ringu – 770mm,
wysokość ringu – 25mm,
szerokość linii brzegowej – 25mm,
szerokość linii startowej – 10mm,
długość linii startowej – 100mm,
odległość linii startowej od środka – 50mm,
promień obszaru zewnętrznego ringu – 138mm (chodzi tu o obszar, gdzie nie powinno się stać podczas walki).

Wygląd takiego dojo widać na **fotografii 1**. Na zawodach podstawową zasadą jest dobra zabawa i fair play. Fajny opis zasad minisumo znajduje się pod adresem: **http://roboty.utp.edu.pl/index.php?option=com_content&task=view&id=28&Itemid=70**.

„INTELIGENCJA" robota

Przez kilka ostatnich odcinków starałem się wytłumaczyć zasadzę działania programów pisanych w języku C. Mam nadzieję, że skutecznie, gdyż kwestie programowe nie były poruszane w e-mailach od Was. W tej części zaprezentuję prosty kompletny program sterujący robotem, który, mam nadzieję, będzie rozbudowywany i przerabiany.

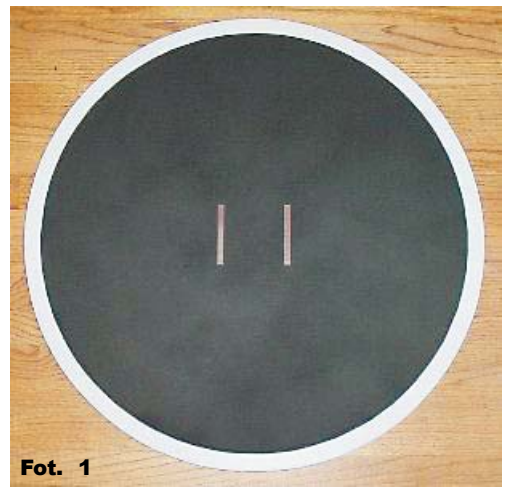
Program, po pierwsze, musi sterować silnikami; po drugie, musi sprawdzać stan czujników koloru; po trzecie, musi w razie aktywacji wąsów wykonać jakąś czynność; po czwarte, warto zastosować PWM, by robot nie „szalał na dojo".

Pierwszą sprawą jest odpowiednie zainicjowanie portów IO mikrokontrolera, do czego, jak wiemy, służą rejestry DDRx (gdzie x to nazwa portu). Następnie trzeba uruchomić obsługę PWM i przerwania.

Aktywacją portów zajmuje się procedura Init_IO_port(); – **listing 1**. Wszystkie powyższe rejestry bardzo dokładnie opisane są w nocie katalogowej mikrokontrolera ATMEGA16/32.

Aby robot zadziałał, należy uruchomić procedurę, która aktywuje silniki, ale jednocześnie trzeba pamiętać, że robot może wypaść z ringu, jeżeli nie zobaczy białej linii brzegowej. Toteż pokusiłem się o uruchomienie obsługi przerwania od timera, by niezależnie od działania robota, czujniki linii były monitorowane „ciągle". Służy do tego procedura Init_TIMER(); – **listing 2**.

Wykorzystuję TIMER0, a dokładniej przerwanie od jego przepełnienia… zapraszam do czytania noty katalogowej. Włączony jest tu tzw. prescaler z wartością 64. A to oznacza, że 64 takty zegara generują zwiększenie się licznika TIMER0 o 1. Jeżeli wartość TIMER0 przekroczy wartość 255, to generowane jest przerwanie i licznik znów liczy od 0 do 255. Mamy zegar 8MHz, dzielimy go przez 64 = 125 000kHz, dalej dzielimy przez 256, co daje około 490Hz… dzięki czemu sprawdzanie czujników jest na tyle szybkie, że nasz robot nie spadnie z ringu. Aby uruchomić system przerwań, należy wywołać takie polecenie:



Fot. 1

Listing 2

```
void Init_TIMER(void)
{
TCCR0=0x03;       // prescaler 64
TCNT0=0x00;       // wartośc początkowa zliczania = 0
TIMSK=0x01;    // Timer/Counter0 Overflow interrupt is enable
}
```

Listing 1

```
Init_IO_port(void)
{

}

, obsługą PWM zajmuje się procedura Init_PWM();
void init_PWM(void)
{
// Ustawiam bity zgodnie ze specyfikacja atmegi */
// FAST PWM with pins OC1A and OC1B,
// Counter1 Prescaler 8,
// Mode 5 - Fast PWM, 8bit, TOP=0x00FF,
// Set OC1A/OC1B on Compare Match, Clear at TOP
TCCR1A|=_BV(COM1A1);
TCCR1A|=_BV(COM1B1);
TCCR1A|=_BV(WGM10);
TCCR1B|=_BV(WGM12);
TCCR1B|=_BV(CS11);
OCR1AH=0x00;  //TEN REJESTR NIE MA W ZASADZIE ZNACZENIA I TAK PWM PRACUJE 8 BITOWO
OCR1AL=0;
OCR1BH=0x00;  //TEN REJESTR NIE MA W ZASADZIE ZNACZENIA I TAK PWM PRACUJE 8 BITOWO
OCR1BL=0;
}
```

__asm(„sei"); //zezwolenie na przerwania globalne.

Aby robot zrobił co do niego należy, trzeba dodać procedurę wywoływaną za pomocą TIMER0. Tutaj nie możemy sobie pozwolić na dowolność w nazewnictwie, gdyż środowisko programistyczne wymusza na nas określone nazwy i tak dla TIMER0 procedura obsługi przerwania Overflow nazywa się *SIGNAL (SIG_OVERFLOW0)*. Jak się okazuje, każde środowisko programistyczne ma własne nazwy procedur obsługi przerwań. Trzeba o tym pamiętać, jeśli chcemy się uczyć na gotowych przykładach pochodzących z AVRGCC, CODEVISION czy mnóstwa innych. Ważna wskazówka – jeżeli obsługujemy przerwania, to sugeruję, by na czas wykonywania obsługi przerwania wyłączać możliwość wykrywania innych przerwań. Oczywiście nie jest to konieczne, a czasami wręcz nie wolno tak zrobić, ale należy pamiętać o tym, że procesor ma określoną liczbę zapętleń i po prostu może nam narobić bigosu i w końcu się zawiesi. Pisząc program warto się nad tą kwestią zastanowić.

Wyłączenie obsługi przerwań polega na

wywołaniu polecenia *__asm(„cli”);* A na końcu procedury obsługi przerwania należy ponownie je uruchomić, czyli procedura obsługi przerwań w naszym przypadku będzie wyglądać jak na **listingu 3**.

PROGRAM GŁÓWNY

Mam nadzieję, że program będzie dla Ciebie jasny. **Ściągnij go z Elportalu** (**listing 4**). Program zaczyna się w pętli main(). Pierwsze, na co trafia nasz program, to polecenie Init_IO_port();, dalej aktywujemy wyjścia PWM oraz ustawiamy parametry TIMER0. Później jest pętla, czekająca na naciśnięcie przycisku START. Jest to wymagane w klasie minisumo. Po naciśnięciu przycisku program wywołuje procedurę migaj_led(4); jej zadaniem jest odmierzenie 5 sekund od wciśnięcia przycisku. Jest to także czas wymagany przez regulamin minisumo, co dodatkowo jest sygnalizowane mignięciami diody LED. W czasie tych 5 sekund należy odsunąć się od robota. Po odliczeniu 5 sekund zaczyna się „walka”. Następne polecenie aktywuje przerwania. I ostatnie polecenie – pętla wywołująca procedurę *szukaj*. Pętla ta wykonywana jest w zasadzie w nieskończoność. Przerywana jest działaniem przerwania od TIMER0.

Procedura *szukaj* określa sposób przemieszczania się robota po dojo. W wersji bezczujnikowej w zasadzie nie ma to większego sensu, ale wygląda atrakcyjnej niż robot stojący w miejscu. Sposób przemieszczania się robota określony jest kilkoma zmianami kierunku, w zależności od odliczonej wartości zmiennych *licznik* i *kierunek*.

Procedury *stoj, doprzodu, dotylu LEWY(PRAWY)* służą do uruchomienia silników. Każda z tych procedur ma parametr, jakim jest szybkość (prędkość). Parametr ten to wartość wpisywana do rejestru PWM i powoduje ona zmianę współczynnika wypełnienia przebiegu, za czym idzie zmiana wartości średniej napięcia (prądu) płynącego przez silnik. Skutkiem tego są zmiany prędkości obrotowej kół, przydatne przy wybieraniu ataku, ucieczki czy szukania.

Oczywiście jest to tylko szablon, a każdy z Was może sprawić, by jego robot był unikatem. Nie oczekuję „wysypu” klonów robiących dokładnie to samo.

Jednakże należy od czegoś zacząć. Bazą dla mojego programu było kilkanaście programów, które przeanalizowałem i doprowadziło to do powstania swego rodzaju kompilacji. Czy jest dobra, czy nie? To kwestia sporna. Działa i to najważniejsze. Sugeruję byś samodzielnie zmodyfikował zachowanie Twojego pupila, tak jak Tobie się spodoba.

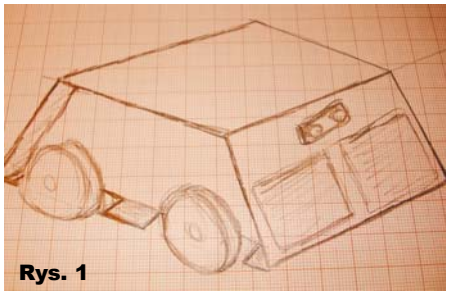
Drogi Czytelniku, wbrew pozorom całe to programowanie jest proste, wymaga jednak sporo ćwiczeń i kombinowania, a reszta przychodzi sama z czasem. Po tym kursie będziesz w stanie samodzielnie pisać programy nie tylko dla tego robota.

```
SIGNAL (SIG_OVERFLOW0)
{
__asm(„cli”);

//zestaw poleceń do wykonania

__asm(„sei”);
}
```

Listing 3



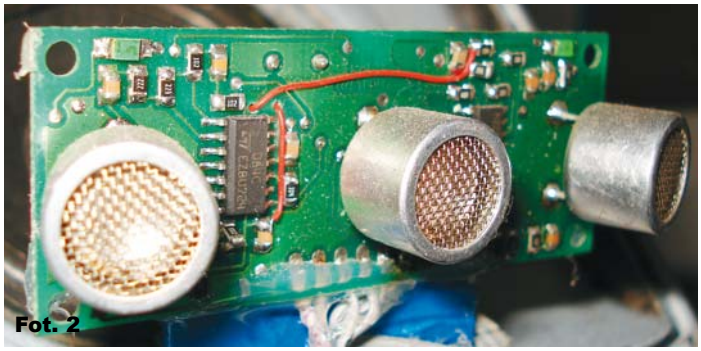
Rys. 1

Podsumowanie

Robot jest prostą konstrukcją, która ma wnieść trochę radości i optymizmu w życie ludzi, którzy chcą zacząć z nim swoją przygodę. Robot docelowo na pewno będzie wymagał dobudowania czujników wykrywających przeciwnika lub przeszkody z odległości. Czujniki, „wąsy” dotykowe podstawowej wersji okażą się niewystarczające. Idea późniejszej rozbudowy robota polega na dołożeniu bezdotykowego, ultradźwiękowego czujnika odległości w procedurze obsługi przerwania. Takie czujniki dają informację, że na dojo znajduje się przeciwnik w odległości x od naszego. Natomiast nasze czujniki dotykowe muszą wykryć fizyczny kontakt z przeciwnikiem. Czy to dobrze, czy źle? Do nauki podstaw dobrze. Do walki w ringu… źle. Ale nie przejmuj się, Drogi Czytelniku. Płytka mózgu robota jest wyposażona w wejścia umożliwiające podłączenie różnych czujników. A zmiany programowe są proste.

Na rynku, obok gotowych modułów firmy Sharp, dostępne są m.in. czujniki z KONAR-u (**fotografia 2**), dokumentacja do ich wykonania znajduje się na stronie: **http://www.konar.pwr.wroc.pl/uploads/download/raporty/sonar.pdf** Czujnik KONAR-u zastosowany jest w robocie minisumo z fotografii tytułowej. Następne są czujniki dostępne na stronie **www.mobot.pl** (**fotografie 3** i **4**). Bez takich „sprytniejszych” czujników nasz robot to tylko platforma do przeprowadzania doświadczeń i nauki programowania.

Na **rysunku 1** pokazana jest bardzo uproszczona koncepcja. Mam nadzieję, że już w następnym odcinku uda mi się przedstawić Ci zrealizowany model robota minisumo. Napęd będą stanowiły 4 silniki z przekładniami. Na tym płytki, czyli nasza kanapka z elektroniką. I mamy gotowego robota. Reszta to już tylko unowocześnianie konstrukcji, dodawanie bajerów i zmiany w programie, by robot był bardziej inteligentny.



Fot. 2

Fot. 3

Fot. 4

Sądzę też, że spotkamy się kiedyś na zawodach, których w Polsce jest coraz więcej.

Na koniec mała uwaga praktyczna: jeżeli ktoś z Czytelników ma kłopot z układem L293D (nie chce działać), może to być spowodowane zbyt niskim napięciem zasilania. Układ stabilizatora powinien dostarczać minimum 4,5V (5V zalecane), a podczas moich zabaw okazało się, że nie każdy L293D chciał działać przy napięciu 3,3V (a takie testy też robiłem). W niektórych sklepach sprzedawcy nie patrzą, jakie napięcie stabilizuje LM1117, a są różne wersje tego układu.

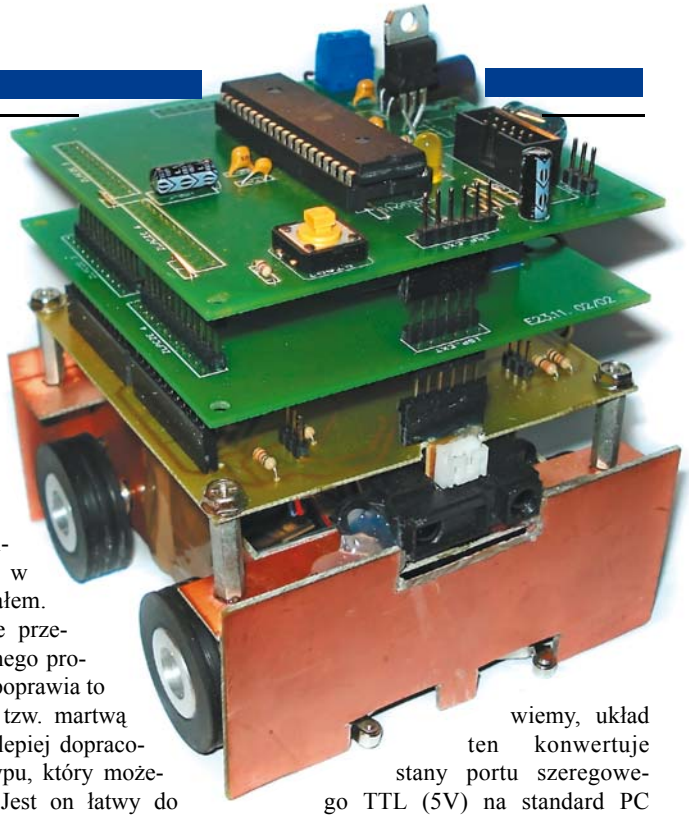
Drugą sprawą jest to, że na czas programowania należy zdjąć zworkę, znajdującą się na płytce „mózgu” robota. A na czas pracy znów ją założyć. Obecność tej zwory może zablokować możliwość programowania mikrokontrolera lub błędy podczas programowania. Jest to zjawisko normalne, nie należy się nim przejmować, tylko pamiętać o zdejmowaniu tej zworki.

Pozdrawiam i zapraszam do zadawania pytań, dotyczących dalszego rozwoju cyklu.

Marek Majewski
architectus21st@gmail.com
office@inventco.eu

Robot mobilny – krok po kroku

część 8

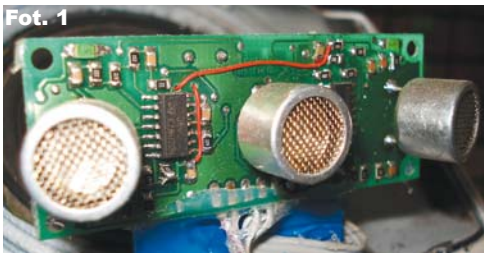


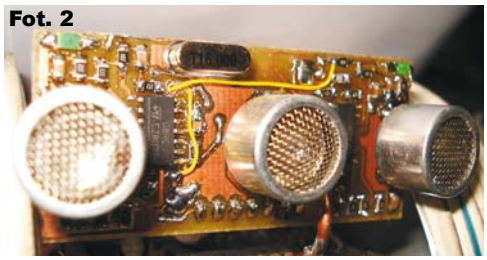
W ostatniej części cyklu zajmiemy się omówieniem dodatkowych czujników, które można zastosować w naszej konstrukcji. Jeden wymaga zrobienia, drugi kupienia. Zapraszam do czytania.

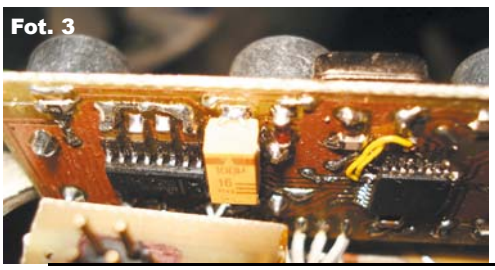
Czujniki, czujniki – co zrobić, by robot był bardziej interaktywny?

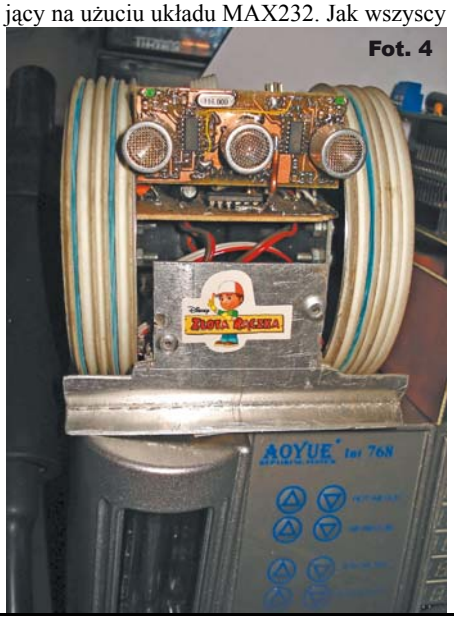
Pierwszą ważną rzeczą jest to, by robot mógł zrobić coś więcej, niż tylko jechać do przodu i odbijać się od każdej białej linii. Ewentualnie jeśli trafi przez przypadek na przeciwnika, to dopiero wtedy ruszy do ataku. Niezmiernie trudno jest trafić bardziej zaawansowanego przeciwnika tak, by nasz robot mógł z nim współzawodniczyć. Dlaczego?... z prostej przyczyny: każdy robot na ringu stara się także wygrać z naszym, poprzez odpowiednie algorytmy i/lub zaawansowany system czujników. Opisywany robot jest tylko platformą przeznaczoną do dalszej rozbudowy. Warto zainteresować się przede wszystkim czujnikiem, o którym już kilka razy wspominałem. Nie jest to moje opracowanie, ale jest dostępne za darmo w Internecie. Mowa tu oczywiście o stereofonicznym czujniku sonarowym, opracowanym przez studentów Koła Naukowego Robotyków KONAR z Wrocławia. Cała potrzebna dokumentacja znajduje się na stronie **www.konar.pwr.wroc.pl**. Na **fotografii 1** widoczny jest oryginalny czujnik opracowany właśnie przez tę organizację. Płytkę otrzymałem w prezencie i ją zmontowałem. Widoczne dwa dodatkowe przewody służą do „dynamicznego progowania" wejść czujnika (poprawia to selektywność i zmniejsza tzw. martwą strefę). Jest to jeden z najlepiej dopracowanych czujników tego typu, który możemy zrobić samodzielnie. Jest on łatwy do wykonania i nie sprawia kłopotów przy uruchamianiu. Na **fotografiach 2** i **3** widoczny jest ten sam czujnik, ale na zmodyfikowanej przeze mnie płytce drukowanej. Modyfikacja polegała na przesunięciu elementów bliżej środka i skróceniu płytki po to, by zmieściła się w moim robocie minisumo (**fotografia 4**). Nawet tak radykalne posunięcia nie spowodowały wadliwej pracy czujnika.

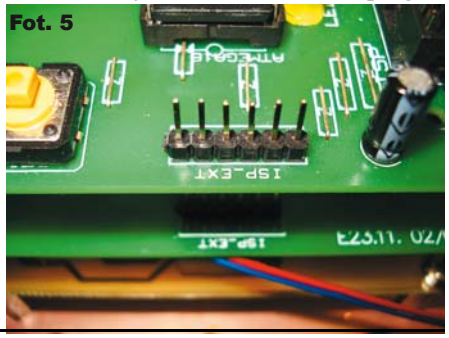
A dlaczego czujnik sonarowy? Jest on dość rzadko stosowany na zawodach i moim zdaniem trudniejszy do oszukania. Czujniki działające na podczerwień są doskonale oszukiwane poprzez specjalne rodzaje farb pochłaniające promieniowanie i są coraz częściej stosowane na zawodach w bardziej zaawansowanych konstrukcjach. Czujnik KONAR-u ma sterownik oparty o mikrokontroler ATmega48 – steruje on zarówno odbiorem, jak i nadawaniem. Do wytworzenia wyższego napięcia na przetworniku nadawczym, studenci wykorzystali bardzo prosty, acz genialny trik, polegający na użuciu układu MAX232. Jak wszyscy wiemy, układ ten konwertuje stany portu szeregowego TTL (5V) na standard PC RS (+/-12V). Dzięki temu można uzyskać amplitudę rzędu 24V bez konieczności stosowania przetwornic i elementów indukcyjnych. Obwody wejściowe zrealizowane zostały na popularnym wzmacniaczu operacyjnym LM324. Oczywiście wszystkie te elementy są montowane powierzchniowo w celu zmniejszenia wymiarów modułu. Zasada działania czujnika sonarowego jest prosta: element (przetwornik) nadawczy wysyła w świat paczkę piknięć (częstotliwość to około 40kHz), po czym te „piknięcia", docierając do przeszkody, odbijają się i docierają do odbiorników. Tutaj są wzmacniane i kształtowane tak, by mikrokontroler mógł zmierzyć czas pomiędzy wysłaniem a odebraniem, co po przeliczeniu za pomocą podstawowych wzorów z fizyki daje nam informację, w jakiej odległości znajduje się przeszkoda. Czujnik opracowany przez członków KONAR-u podaje prawidłową odległość do około 80cm (czyli w zasadzie pokrywa całą powierzchnię dojo). Ja osobiście zrobiłem progowanie na jakieś 15–20cm od robota, gdyż uważam, że nie ma sensu więcej. Ale to zależy od Ciebie, Drogi Czytelniku. Sonar ten komunikuje się z naszym robotem za pomocą portu SPI (to te kilka szpilek specjalnie wyprowadzonych na przedniej części modułu „mózgu" – widoczne na **fotografii 5**. Po zaprogramowaniu i podłączeniu płytek powinno wszystko zadziałać od razu. Tutaj **UWAGA!!! Podczas progra-**


Fot. 1

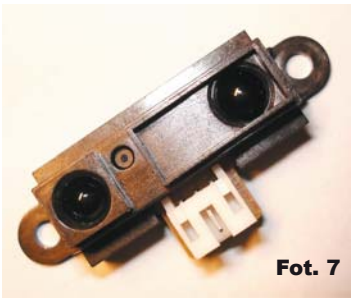

Fot. 2

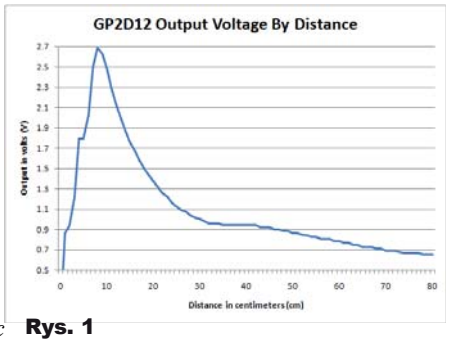

Fot. 3


Fot. 4


Fot. 5

Fot. 6


Fot. 7



Rys. 1

mowania robota należy bezwzględnie odłączać moduł sonaru od płytek robota, inaczej możemy go uszkodzić. To dość istotna uwaga, gdyż sam kilkanaście razy musiałem programować moduł sonaru przez ten błąd. Po krótkim omówieniu zalet tego rozwiązania warto wspomnieć, jak go użyć. Otóż trzeba za pomocą poleceń *#include* dodać do programu biblioteki niezbędne do jego działania, czyli na początku programu, tam gdzie są umieszczone pozostałe „załączniki", należy dodać wpis: *#include „SPI.h"* i upewnić się, że pliki SPI.h i SPI.c znajdują się w katalogu naszego programu. Pliki te do pobrania są oczywiście na stronie KONAR-u. Dodatkowo należy wprowadzić następującą sekwencję w „main":

```
Inicjalizacja_SPI_Master();
_delay_ms(4000);
wersja = Inicjuj_sonar();
```

Czas 4s nie jest tu wartością krytyczną – po prostu ją wybrałem, tak wpisałem i działa, ale możesz poeksperymentować z krótszymi czasami. Ta sekwencja powoduje zainicjowanie sonaru i odczytanie wersji programu. Dopiero po takiej sekwencji sonar jest gotów do pracy. Oczywiście należy zainicjować odpowiednio porty i rejestry w „mózgu" naszego robota, tak aby uaktywnić procesor do pracy w trybie SPI – wykonuje to polecenie *Inicjalizacja_SPI_Master();*. Gdy już wszystko zadziała, objawi się to ciągłym świeceniem dwóch diod LED na płytce sonaru. W pętli programu należy użyć poleceń „mierzących" wartości: *Zrob_Pomiar(1,&Lewy, &Prawy);*. Oczywiście w programie należy zadeklarować wartości zmiennych potrzebnych do poprawnej pracy, czyli:

```
unsigned char wersja = 0;
unsigned int Lewy = 0, Prawy = 0;
```

Z Elportalu można ściągnąć program robota minisumo. Zawiera on obsługę sonaru opracowanego przez KONAR. Tu uwaga: nie jest to gotowe rozwiązanie – należy je dostosować do naszego robota, przypisać odpowiednie wejścia, itp. Program działa, jest sprawdzony. Jest to oczywiście przykład, ale w zupełności wystarczający dla każdego, kto chce powalczyć. W programie zobaczycie, że wymaga on pliku defines.h. Zawartość tego pliku znajduje się dalej:

```
//tu mają byc wszystkie definicje
/*============Definicje dla kompilatora==========*/
#define cbi(add,bit) ((add) &= ~(1 << bit));
#define sbi(add,bit) ((add) |= (1 << bit));
#define F_CPU 8000000UL //predkosc taktowania
/*============Definicje dla sonaru==========*/
#define Nop 0xF0
#define Liczba_bajtow_inicjujacych 4
#define Najdluzsza_ramka 4
```

Bez tego pliku lub tych informacji dodanych do programu sonar nie będzie działał, a program będzie wyrzucał błędy kompilacji. Część tych poleceń już widzieliście nieraz, ale część jest nowa. Ta nowa część dotyczy tylko i wyłącznie potrzeb sonaru. Życzę powodzenia w eksperymentowaniu!

To tyle, jeśli chodzi o sonar. Reszty informacji dostarczy bardzo ładnie napisany PDF z precyzyjnym opisem sonaru, znajdujący się na ww. stronie www. Do pobrania są tam także rysunki płytek drukowanych i opis przeróbki na wersję 4 (tę z dynamicznym progowaniem).

Czujniki firmy SHARP

Teraz omówimy czujniki firmy SHARP, GP2D12 (**fotografie 6** i **7**). Są różne opinie na ich temat. Ale to nie zmienia faktu, że są to jedne z lepszych i łatwo dostępnych rozwiązań. Mają jednak wadę, moim zdaniem dużą – jest nią nieliniowa charakterystyka odzwierciedlająca odległość od mierzonej przeszkody (**rysunek 1**). Zaraz ktoś powie, że można ją „zmapować" lub za pomocą obliczeń matematycznych skorygować. Owszem, można, ale fabrycznie zlinearyzowany czujnik byłby lepszy w obsłudze.

Oczywiście można ten czujnik wykorzystać jako wskaźnik obecności przeciwnika, by nasz robot wiedział, że przed nim coś się znajduje i mógł podążyć w kierunku celu. Podpięcie tego czujnika jest banalne. Potrzebuje on bowiem zasilania 5V i masy (które mamy na płytce robota), prócz tego wyjście czujnika należy podłączyć do dowolnego wolnego wejścia PORTUA naszej ATmegi i oprogramować (to wejście) jako ADC. Następnie w programie należy okresowo sprawdzać wartość, jaką zwraca przetwornik ADC z tego wejścia. I odpowiednio do tego, za pomocą poleceń *if* bądź *case*, wybierać działanie robota. Ot i cała filozofia. Osobiście uważam, że czujniki te są dobre, ale tylko wtedy, gdy zastosujemy ich kilka, co najmniej dwa z przodu. Dlaczego? Otóż mamy informację, z której strony znajduje się przeciwnik. Należy bowiem pamiętać, że przeciwnik także znajduje się w ruchu. Nie będę tutaj rozpisywał się, jak oprogramować ten czujnik, gdyż na podstawie całego kursu większość czytelników powinna potrafić samodzielnie już coś takiego zrealizować.

Zakończenie – słów kilka...

Bardzo się cieszę, że mogłem się z Wami, podzielić moją wiedzą na temat robotów i mikrokontrolerów. Mam nadzieję, że pomoże i zachęci do eksperymentowania w tej dziedzinie, gdyż możliwości są przeogromne. Mam nadzieję, że także i Wy pokochacie tę dziedzinę elektroniki, a właściwie to już mechatroniki.

Pozdrawiam, życzę wielu sukcesów i jak zwykle zachęcam do pisania do mnie e-maili.

A jeśli chcecie, by cykl był kontynuowany, wykorzystajcie Miniankietę EdW i piszcie e-maile do redakcji (*redakcja@elportal.pl*).

Marek Majewski
architectus21st@gmail.com
office@inventco.eu